No. 16-2417

UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

APPISTRY, LLC,
Plaintiff-Appellant,
v.

AMAZON.COM, INC. AND AMAZON WEB SERVICES, INC.,
Defendants-Appellees.

Appeal from the United States District Court for the Western District of Washington at Seattle in Case No. 2:15-CV-01416-RAJ, Judge Richard A. Jones

CORRECTED PLAINTIFF-APPELLANT APPISTRY, LLC'S OPENING BRIEF

DATED: October 4, 2016

Anthony G. Simon
Timothy D. Krieger
Benjamin R. Askew
Michael P. Kella
THE SIMON LAW FIRM, P.C.
800 Market Street, Suite 1700
St. Louis, MO 63101
Telephone: (314) 241-2929
Facsimile: (314) 241-2029
asimon@simonlawpc.com
tkrieger@simonlawpc.com
baskew@simonlawpc.com
mkella@simonlawpc.com

Attorneys for Plaintiff-Appellant Appistry, LLC

CERTIFICATE OF INTEREST

Counsel for Plaintiff-Appellant Appistry, LLC certifies the following:

1. The full name of every party or amicus represented by me is: Appistry, LLC.

2. The name of the real party in interest (Please only include any real party in interest NOT identified in Question 3, below) represented is: Appistry, LLC.

3. All parent corporations and any publicly held companies that own 10 percent of the stock of the party or amicus curiae represented by me are listed below. (Please list each party or amicus curiae represented with the parent or publicly held company that owns 10 percent or more so they are distinguished separately.) None.

4. The names of all law firms and the partners and associates that appeared for the party or amicus now represented by me in the trial court or agency or are expected to appear in this court (and who have not or will not enter an appearance in this case) are:

Law Firm	Attorneys
Lane Powell PC 1420 Fifth Avenue, Suite 4200 P.O. Box 91302 Seattle, Washington 98111-9402 Telephone: (206) 223-7000 Facsimile: (206) 223-7107	Brian G. Bodine Adriane M. Scola

Law Firm	**Attorneys**
Haar & Woods, LLP 1010 Market St., Suite 1620 St. Louis, Missouri 63101 Telephone: (314) 241-2224 Facsimile: (314) 241-2227	Robert T. Haar Colleen O. Zern

DATED: October 4, 2016

By: *<u>/s/ Anthony G. Simon</u>*

Anthony G. Simon
Timothy D. Krieger
Benjamin R. Askew
Michael P. Kella
THE SIMON LAW FIRM, P.C.
800 Market Street, Suite 1700
Saint Louis, Missouri 63101
P. 314.241.2929
F. 314.241.2029

Attorneys for Plaintiff-Appellant
Appistry, LLC

TABLE OF CONTENTS

TABLE OF AUTHORITIES

STATEMENT OF RELATED CASES

Pursuant to Fed. Cir. Rule 47.5(a), no other appeals in or from the same civil action or proceeding were previously before this or any other appellate court.

Pursuant to Fed. Cir. R. 47.5(b), this Court has ordered that this case and *Appistry, LLC v. Amazon.com, Inc. & Amazon Web Services, Inc.*, Case No. 2015-2077 (Appeal from the United States District Court for the Western District of Washington in No. 2:15-cv-00311-MJP)("Appistry I") will be treated as companion cases and will be assigned to the same merits panel.

JURISDICTIONAL STATEMENT

This appeal arises from a decision of the U.S. District Court for the Western District of Washington holding that Appistry's patents were invalid under 35 U.S.C. § 101. The District Court had jurisdiction under 28 U.S.C. §§ 1331 and 1338(a). This Court has jurisdiction under 28 U.S.C. § 1295. Final judgment was entered on July 27, 2016. A notice of appeal was filed on July 28, 2016, within the timeframe of 30 days as required by Fed. R. App. P. 4(a)(1)(A).

STATEMENT OF ISSUES

1. Did the Washington District Court err in granting Amazon's Motion to Dismiss for Invalidity Under 35 U.S.C. § 101 finding that the claims of U.S. Patent Nos. 8,682,959[1] ("'959 Patent) and 9,049,267 ("'267 Patent) (hereinafter "Appistry Patents") recite patent-ineligible subject matter?

STATEMENT OF THE CASE

On September 3, 2015, Appistry filed a Complaint in the Western District of Washington alleging patent infringement by Amazon. On October 15, 2015, Amazon filed a Motion to Dismiss for Invalidity Under 35 U.S.C. § 101. On July 19, 2016, Amazon's Motion was granted finding that "the claims of the '959 and '267 Patents are directed to an abstract idea" and "do not contain an inventive concept."

[1] As the specifications of the '959 Patent and the '267 Patent are substantially similar, for convenience, citations will generally be limited to the '959 Patent.

STATEMENT OF FACTS

Appistry, founded in St. Louis, Missouri in 2001, developed a specific implementation of a distributed computing system that is protected by the Appistry Patents. Appx121, ¶ 9. Appistry's award-winning "fabric" distributed computing system was a breakthrough in facilitating multiple computers to work together to increase processing power, reliability, and scalability. *Id.*; Appx129, ¶¶ 60, 66. Appistry's distributed computing system is used by dozens of companies in areas such as intelligence, defense, life sciences, financial services, and transportation. Appx121-122, ¶ 10.

I. The Field of the Invention: Distributed Computing

The inventions in the Appistry Patents were created to address the growing need for reliable and affordable processing power. Appx053: Col. 1, Lines 44-51. As of the filing date of the patents, there were five alternative platforms: mainframes, high-availability computers, UNIX-based servers, distributed supercomputers, and personal computers. Appx053: Col. 1, Lines 59-63. Each platform had shortcomings. Appx053: Col. 1, Lines 63-65. Mainframes were expensive and not maintainable, requiring numerous highly-trained engineers and custom hardware. Appx053: Col. 2, Lines 18-21. As a single-box machine, mainframes were inevitably vulnerable to single-point failures. Appx053: Col. 2, Lines 6-8. High-availability computers had many of the same shortcomings as mainframes. Appx053: Col 2, Lines 22-41. UNIX-based servers required application developers to be experts in UNIX and the application. Appx053: Col. 2, Lines 62-67; Appx054: Col. 3, Lines 1-17. Distributed super computers utilized the resources of underused desktop computers under the control of different owners, making this platform unsuitable for a business' critical requirements. Appx054: Col. 3, Lines 42-59. Finally, personal computers were designed as stand-alone machines and lacked the processing capacity. Appx054: Col. 4, Lines 1-18.

II. The Appistry Patents

To address these issues, Appistry invented a specific distributed computer system that vastly increased processing capabilities and added reliability and scalability. Appx055: Col. 5, Lines 18-22. The '959 Patent has a priority date of September 7, 2002. Appx026. The '959 Patent teaches configuring multiple "hive engines" on a series of networked computers to create processing power. Appx055: Col. 5, Lines 18-27; Appx066: Col. 27, Lines 49-54.

The '267 Patent is a continuation of the '959 Patent. Appx070. The Appistry Patents describe with particularity a configuration of components spread across multiple computers. *See, e.g.*, Appx066: Col. 27, Lines 51-67; Col. 28, Lines 1-11. These components include request handlers, process handlers and task handlers. *Id*. As set forth in Claim 1 of the '959 Patent, a request handler receives a service request for processing a job and communicates job information to a process handler. Appx066: Col. 27, Lines 59-61. The process handler receives data for the processing job, identifies the tasks to be performed and sends the individual tasks to task handlers. Appx066: Col. 27, Lines 62-67; Col. 28, Lines 1-5. The task handlers perform the tasks and generate status updates on the tasks. Appx066: Col. 28, Lines 1-5. The request handler monitors the status of the tasks, determines whether any fault exists, and if so, initiates a recovery procedure based on the status information it maintains. Appx066: Col. 28, Lines 6-11.

The particular functions and configurations of the three tiers of handlers, implemented on a distributed computer system, did not exist prior to the inventions of the Appistry Patents. Appx054: Col. 3, Lines 18-50. The Appistry Patents discuss the relevant prior art as of their priority date. Appx026; Appx070. There are undisputed distinctions between the prior art referenced in the patents and the claimed inventions. Appx054: Col. 3, Lines 18-50. In the prior art, large problems were broken up into smaller tasks, "spread across many small computers, solved **independently**, and then brought back together." Appx054: Col. 3, Lines 18-24. (any emphasis throughout is added unless noted). Among other things, the prior art systems did not compute and distribute information about the status of each processing task because they were independent. Appx054: Col. 3, Lines 18-50.

In contrast, the inventions described in the Appistry Patents set out a particular, and previously unknown, configuration of three tiers of handlers for the purpose of solving an overall problem where tasks are **<u>sequentially dependent</u>** on other tasks in the context of solving the overall problem. Appx055: Col. 5, Lines 49-67; Col. 6, Lines 1-7. Also unique is the ability of the "hive engines" to use this particular pattern to organize *themselves* into logical groups to perform the processing tasks without any prior configuration. Appx058: Col. 11, Lines 49-53. The *self-organization* method guarantees execution of a request even if any individual hive engines fails. Appx058: Col. 12, Lines 57-62. At each discrete step, the status of the overall progress of the job is known by both the request handler and process handler. Appx058: Col. 12, Lines 57-67; Appx059: Col. 13, Lines 1-12.

The use of "hive engines" and the particular configurations of computer components described in the Appistry Patents solve the reliability and scalability problems inherent in prior art. Appx055: Col. 5, Lines 18-22. The distinct pattern of tiered coordination among the components compensates for the failure of any machine on the network. Appx058: Col. 12, Lines 57-67; Appx059: Col. 13, Lines 1-12. It also efficiently allocates resources on the network and allows for the addition of resources to the network in a cost-effective manner. Appx058: Col. 11, Lines 49-53; Appx060: Col. 15, Lines 32-49.

In 2004, Amazon sought to license Appistry's technology. Appx122, ¶ 11. Appistry and Amazon participated in several lengthy meetings. Appx122, ¶¶ 12–16. An initial meeting was held in approximately August of 2004. Appx122, ¶ 12. A second meeting was held on September 14, 2004. Appx122, ¶ 15. Amazon's Director of Systems Research and approximately 10 Amazon senior technical engineers directly involved in Amazon's cloud services participated in the second meeting. Appx122, ¶ 15. In this four hour meeting, Appistry answered numerous, highly detailed questions about the functionality of Appistry's technology. Appx122, ¶ 16. Appistry also, at Amazon's insistence, disclosed very specific algorithms, flow charts, and branches in the decision tree of Appistry's technology with the understanding that Amazon would honor Appistry's patent rights. Appx122-123, ¶ 17–18. Amazon declined to take a license. Appx123, ¶ 21. Years later, Appistry discovered that Amazon copied its technology. Appx123, ¶ 22.

<u>SUMMARY OF ARGUMENT</u>

Regarding the first prong of *Alice,* the District Court erred by finding the claims directed to an abstract idea. The specifications in the Appistry Patents explain that the inventions recited in the claims are directed to an architecture for a new and improved type of distributed computer and explain the shortcomings of prior art distributed computers. Like the claims in *Enfish, LLC v. Microsoft Corp.*, 822 F.3d 1327 (Fed. Cir. 2016), the present claims are directed to specific improvements to the way distributed computers operate, a concrete and tangible result, and are undoubtedly **not** abstract.

Regarding the second prong of *Alice Corp. Pty. Ltd v. CLS Bank Int'l*, 134 S. Ct. 2347, 2355 (2014), which the District Court should not have even reached, the District Court erred by making unsupported findings, which were contrary to the specifications, Complaint, and expert declaration attached to the Complaint, that the computer functions recited in the claims were conventional and that the inventions did not improve the functioning of the computer itself. Unrebutted factual allegations in the pleadings also explain that the computer functions recited in the claims were not performed in prior art distributed computers. Unrebutted statements in the pleadings explain how the inventions recited in the claims, as a whole, improve the functioning over prior art distributed computers. Thus, at a minimum, like the claims in *Diamond v. Diehr*, 450 U.S. 175 (1981), *DDR Holdings, LLC v. Hotels.com, L.P.*, 773 F.3d 1245 (Fed. Cir. 2014) and *Bascom Global Internet Servs. v. AT&T Mobility LLC*, 827 F.3d 1341 (Fed. Cir. 2016), the claims recite patent-eligible applications of any alleged abstract concept.

As the above evidence was unrebutted, the District Court also erred in making unsupported factual findings to the contrary, especially at the Rule 12 stage where all facts recited in the Complaint and its exhibits are taken as true.

Moreover, Amazon did not meets its burden of proving any facts supporting patent-ineligibility by clear and convincing evidence because the facts supporting patent-eligibility are unrebutted. For these reasons, the judgment of the District Court should be reversed.

<u>ARGUMENT</u>

I. The District Court Erred in Granting Amazon's Motion to Dismiss.

A. Standard of Review

In reviewing a grant of a motion to dismiss, this Court applies the procedural law of the regional circuit. *Bascom*, 827 F.3d at 1347. The Ninth Circuit reviews a grant of a Rule 12(b)(6) motion *de novo*. *In re NVIDIA Corp. Sec. Lit*., 768 F.3d 1046, 1051 (9th Cir. 2014). "A complaint must not be dismissed unless it appears beyond doubt that the plaintiff can prove no set of facts in support of the claim that would entitle the plaintiff to relief." *Aguayo v. U.S. Bank*, 653 F.3d 912, 917 (9th Cir. 2011). All allegations of material fact in the complaint are taken as true and construed in the light most favorable to the non-moving party. *Id.* This Court reviews a district court's determination of patent eligibility under 35 U.S.C. § 101 *de novo*. *Bascom*, 827 F.3d at 1347.

B. The *Alice* Test for Patent Eligibility

Patent eligible subject matter includes "any new and useful process, machine, manufacture, or composition of matter or any new and useful improvement thereof." 35 U.S.C. §101. Judicial exceptions to patent eligibility are "[l]aws of nature, natural phenomena and abstract ideas." *Bascom*, 827 F.3d at 1347. In *Alice Corp. Pty. Ltd v. CLS Bank Int'l*, 134 S.Ct. 2347, 2355 (2014), the Supreme Court reaffirmed the two-step analytical framework for determining patent eligibility. First, "the claims are considered in their entirety to ascertain whether their character as a whole is directed to excluded subject matter." *McRO, Inc. v. Bandai Namco Games Am. Inc.*, No. 2016-1080, 2016 WL 4896481, at *6 (Fed. Cir. Sept. 13, 2016) (internal quotations omitted). If the claims are not directed to an abstract idea, the claims are patent-eligible under §101. *Id.* If the claims are directed to an abstract idea, a court must then "look to both the claim as a whole and the individual claim elements to determine whether the claims contain an element or combination of elements that is sufficient to ensure that the patent in practice amounts to significantly more than a patent upon the ineligible concept itself." *Id.* at *7 (internal quotations omitted).

C. The District Court Erred in Holding That the Claims Were Directed to an Abstract Idea.

The first step of the two-stage *Alice* inquiry "is a meaningful one, i.e., [ ] a substantial class of claims are *not* directed to a patent-ineligible concept." *Enfish*, 822 F.3d at 1335. (emphasis in original). For computer-related technology, the first step asks whether the claims, when "considered in light of the specification [and] based on their character as a whole," are "directed to an improvement to computer functionality . . . or, instead on a process that qualifies as an 'abstract idea' for which computers are invoked merely as a tool." *Id.* at 1335–36. The first step "cannot simply ask whether the claims *involve* a patent-ineligible concept, because essentially every routinely patent-eligible claim involving physical products and actions *involves* a law of nature and/or natural phenomenon—after all, they take place in the physical world." *Id.* at 1335. (emphasis in original).

In *Enfish*, this Court found an invention claiming a self-referential table for a computer database was not directed to an abstract idea because the claims were "directed to a specific implementation of a solution to a problem in the software arts." *Id.* at 1339. There, the district court granted summary judgment, finding that the claims were invalid because they were directed to "the concept of organizing information using tabular formats." *Id.* at 1337. In reversing, this Court held that "the district court oversimplified the self-referential component of the claims and downplayed the invention's benefits." *Id.* at 1338. The Court admonished that "describing the claims at such a high level of abstraction and untethered from the language of the claims all but ensures that the exceptions to § 101 swallow the rule." *Id.*

In concluding that the claims at issue in *Enfish* were "directed to a specific improvement to the way computers operate," this Court looked to the specification to determine how the claimed self-referential table functioned differently than conventional database structures. *Id.* at 1336–37. This Court's "conclusion that the claims [were] directed to an improvement of an existing technology [was] bolstered by the specification's teachings that the claimed invention achieves other benefits over conventional databases, such as increased flexibility, faster search times, and smaller memory requirements." *Id.* at 1337; *see also McRO*, 2016 WL 4896481, *6 (relying on specification to confirm claimed improvements over prior art).

The holding in *Enfish* confirms that improvements to computer functionality can be defined by logical processes rather than particular physical features. This Court noted that "[s]oftware can make non-abstract improvements to computer technology just as hardware improvements can." *Enfish*, 822 F.3d at 1335. This Court explained that "[m]uch of the advancement made in computer technology consists of improvements to software that, by their very nature, may not be defined by particular physical features but rather by logical structures and processes." *Id.* at 1339. Accordingly, there is no "exclusion to patenting this large field of technological progress." *Id*.

The District Court in the present case found that the claims of the Appistry Patents were directed to "the abstract idea of distributed processing akin to the military's command and control system." Appx005-006. Much like the district court in *Enfish*, the District Court erred by ignoring the specification, viewing the claims at a high level of abstraction and untethered from the language of the claims, oversimplifying the computer functions recited in the claims, and downplaying the invention's benefits. In fact, the District Court did not cite to *Enfish* in its opinion. Appx002-013.

i. The Claims Are Directed to a New Type Of Distributed Computer Which Improves upon Prior Art Distributed Computing.

In *Enfish,* this Court found the claims at issue "were not directed to an abstract idea within the meaning of *Alice*" because "they [were] directed to a specific improvement to the way computers operate." 822 F.3d at 1336. The Court based its decision, in part, on statements in the specification noting "[t]he specification's disparagement of conventional data structures . . . confirm that our characterization of the 'invention' for purposes of the § 101 analysis has not been deceived by the 'draftsman's art.'" *Id.* at 1339.

As in *Enfish*, the claim language and the specifications in this case explain that "the plain focus of the claims is on an improvement to computer functionality itself, not on economic or other tasks for which a computer is used in its ordinary capacity." *Enfish*, 822 F.3d at 1336. For example, claim 1 of the '959 Patent recites "a plurality of networked computers for processing a processing job in a distributed matter" where those networked computers comprise the following components: "a request handler, a plurality of process handlers, and a plurality of task handlers." Appx066: Col. 27, Lines 51-55. Claim 1 goes on to recite the functions performed by those components. Appx066: Col. 27, Line 59-Col. 28, Line 11.

The Abstract recites the following: "Systems and methods for processing information via networked computers leverage request handlers, process handlers, and task handlers to provide efficient distributed and fault-tolerant processing of processing jobs." Appx026: Abstract. The "Introduction" ends by stating "New mechanisms for computing are desired, especially those which may provide a reliable computing framework and platform, including, but not limited to those which might produce improved levels of performance and reliability at a much lower cost than that of other solutions." Appx055: Col. 5, Lines 18-22. The specification explains that a group of computers, having request handler, process handler, and task handler components distributed among them, each performing specific functions, can work together to provide "vast computing power." *See, e.g.*, Appx055: Col. 5, Line 49-Col. 6, Line 7; Appx060: Col. 15, Lines 32-42. The specification also explains how this configuration provides improved availability, scalability, performance, and reliability. Appx058: Col. 11, Lines 35-48; Col. 12, Lines 14-19; Col. 12, Lines 57-67; Appx059: Col. 13, Lines 1-12; Col. 14, Lines 56-62; Appx060: Col. 16, Lines 56-61; Appx061: Col. 17, Lines 21-25; Col. 18, Lines 44-48; Appx063: Col. 21, Lines 37-51; Col. 22, Lines 24-35.

Similarly, the specification's disparagement of conventional distributed computers confirms that the present claims are directed to improved distributed computers. Appistry's architecture for improved distributed computers has at least two features that were not found in conventional distributed computers: (1) a new, particular configuration of three tiers of handlers for the purpose of solving an overall problem where the tasks are *sequentially dependent* on other tasks in the context of solving the overall problem and (2) the ability of "hive engines" to use this particular pattern to organize *themselves* into logical groups to perform the processing tasks without any prior configuration. Appx055: Col. 5, Line 49-Col. 6, Line 7; Appx057: Col. 10, Lines 52-58. As a result of these features, Appistry's claimed distributed computer provides businesses with a reliable and affordable alternative to conventional distributed computers. Accordingly, the fact "that the claims are directed to an improvement in existing technology is bolstered by the specification's teachings that the claimed invention achieves other benefits over conventional [distributed computers]," such as improved performance, reliability, scalability,[2] and availability.[3] *See Enfish*, 822 F.3d at 1337 (finding "increased flexibility, faster search times, and smaller memory requirements" demonstrated

[2] Scalability is the ability to add additional resources or machines to increase processing power. *See* Appx053: Col. 2, Lines 53-56; Appx054: Col. 3, Lines 32-35.

[3] Appx053: Col. 1, Lines 33-34; Col. 1, Lines 44-50; Col. 2, Lines 13-21; Col. 2, Lines 34-41; Col. 2, Lines 43-46; Appx054: Col. 3, Lines 42-50; Col. 4, Lines 1-10.

improvements over prior art sufficient to support the conclusion that claims were directed to an improvement in computer functionality) (citing *Openwave Sys., Inc. v. Apple, Inc.*, 808 F.3d 509, 513–14 (Fed. Cir. 2015) (holding that a specification's disparagement of the prior art is relevant to determine the scope of the invention)).

The District Court ignored this evidence stating "the claims are not directed to solving a technological problem or solve a challenge particular to a specific environment, nor do they contemplate some 'new' type of computer." Appx008. The District Court's failure to consider the "improvement to computer functionality" provided by the patents-in-suit is directly contrary to this Court's holdings in *Enfish*.

The fact that Appistry's claimed invention allows the use of generic computing components "does not doom[ ] the claims." *See Enfish*, 822 F.3d at 1338. In *McRO, Inc. v. Bandai Namco Games Am. Inc.*, this Court found that claims were "focused on a specific asserted improvement in computer animation, i.e., the automatic use of rules of a particular type." 2016 WL 4896481, at *8. In so holding, this Court stated:

> We disagree with Defendants' argument that the claims simply use a computer as a tool to automate conventional activity. While the rules are embodied in computer software that is processed by general-purpose computers, Defendants provided no evidence that the process previously used by animators is the same as the process required by the claims.

Id. This Court should similarly reject the District Court's finding here that "it is clear that the networked computers utilized in the '959 and '267 Patents are merely generic computers tasked with performing generic functions." Appx007. The specifications of the Appistry Patents and the August 6, 2015 Declaration of Matthew Green, attached as an exhibit to the Complaint, provides evidence that the particular configuration recited in the claims is **not** the same as the configurations used in prior art distributed computers. Appx251-260, ¶¶ 34, 36-53; Appx054: Col. 3, Lines 20-50. In his Declaration, Dr. Green discussed an example of a prior art system, "SETI at Home," and explained that, because state information was not passed in the prior art distributed computer, certain limitations of the claims of the '959 and '267 Patents were not performed by prior art distributed computers. *Id*. The District Court erred by ignoring the facts recited in the Green Declaration and also to the extent that it discounted the Declaration as conclusory. Appx011-012. Dr. Green explained the prior art and then included a limitation-by-limitation analysis of what elements are not disclosed in the prior art. Appx251-260, ¶¶ 34, 36-53. This Court has rejected arguments that such testimony is conclusory. *See, e.g., Commonwealth Sci. & Indus. Research Org. v. Buffalo Tech (USA), Inc.*, 542 F.3d 1363, 1373 (Fed. Cir. 2008).

Moreover, the specification explains that generic computers, on their own, were unable to handle transaction processing and were an undesirable platform for significant business computing tasks. Appx054: Col.4, Lines 1-10 ("Of course, the problem with PC's is that they do not satisfy the needs of businesses and other organizations when it comes to reliability, availability, and predictability."). Accordingly, "[i]t is the incorporation of the claimed rules, not the use of the computer, that improved the existing technological process." *See McRO, Inc.*, 2016 WL 4896481, at *8 (finding rules embodied in software processed by a general-purpose computer asserted specific improvement in technology where "[d]efendants provided no evidence that the process previously used [ ] is the same as the process required by the claims"); *see also Enfish*, 822 F.3d at 1338 (noting that an "invention's ability to run on a general-purpose computer [does not] doom[ ] the claims" where "the claims are directed to an improvement in the functioning of a computer").

Ignoring the facts recited in the pleadings and the language of the claims and specification, the District Court concluded that "the claims were not directed to solving a technological problem or solve a challenge particular to a specific environment" because the claims were not limited to a particular type of task. Appx008. In reaching this conclusion, the District Court failed to account for the difference between the internal operation of Appistry's claimed distributed computer and the tasks that the distributed computer is asked to perform. *See In re Bradley*, 600 F.2d 807, 808 (C.C.P.A. 1979), *aff'd by an equally divided court*, 450

U.S. 381 (1981) (recognizing the difference between claiming internal operation of a computer and applications or tasks the computer is asked to perform). One of the inventive aspects of Appistry's particular configuration for distributed computing is that it is not limited to specific applications or tasks but instead can be used to solve any type of task the user chooses to perform. The District Court misconstrued Appistry's claims and downplayed the benefits of Appistry's distributed computer. *See McRO, Inc.*, 2016 WL 4896481, at *7 ("We have previously cautioned that courts must be careful to avoid oversimplifying the claims by looking at them generally and failing to account for the specific requirements of the claims.") (internal quotation omitted).

In sum, the District Court applied precisely the analysis the *Enfish* Court rejected. It oversimplified the claims and downplayed the inventions of the Appistry Patents. Just like the claims in *Enfish* and *McRO*, the present claims, when tethered to the claim language and viewed in light of the specification, are directed to specific improvements to the way distributed computers operate, a concrete and tangible result, and are "undoubtedly not abstract." *Enfish*, 822 F.3d at 1335; *see also McRO, Inc.*, 2016 WL 4896481, at *10 (finding claims containing "limited rules in a process specifically designed to achieve an improved technological result in conventional industry practice" to not be directed to an abstract idea). Since the claims in the patents-in-suit are not directed to an abstract idea, it is unnecessary to reach step two of *Alice*, and the judgment of the District Court must be reversed.

ii. The Claims Are Not Directed to Producing Intangible Results Such as a Solution to Mathematical Equations or Business Problems

The District Court also erred by not comparing the claims to other claims in previous cases that this Court and the Supreme Court have found to be directed to an abstract idea. The District Court stated that "The Court also rejects Plaintiff's argument that the claims here are not directed to abstract ideas because they do not claim solutions to mathematical questions or business problems." Appx008. But in *Enfish*, this Court noted that "both this court and the Supreme Court have found it sufficient to compare claims at issue to those claims already found to be directed to an abstract idea in previous cases." *Enfish*, 822 F.3d at 1334.

In virtually every case where the Supreme Court has found claims directed to an abstract idea, the patent-ineligible claims were intended to produce an intangible solution to a mathematical equation or business problem. In *Gottschalk v. Benson*, 409 U.S. 63, 65 (1972), the result of the claim was a solution to a mathematical formula: a binary number. In *Parker v. Flook*, 437 U.S. 584, 586 (1978), the result of the claim was a solution to a mathematical formula: an updated alarm limit. In *Bilski v. Kappos*, 561 U.S. 593, 599 (2010), the result of the claim was a solution to a business problem: "how buyers and sellers of commodities in the energy market can protect, or hedge, against the risk of price changes." In *Alice*, the result of the claim was a solution to a business problem: mitigating the risk that only one party to a financial transaction will pay what it owes. *Alice Corp.*, 134 S. Ct. at 2351–52.

In sum, in all of these cases, generic computers were merely used as a tool to perform an underlying application designed to achieve intangible results of a calculation or solution to a business problem. *See Enfish*, 822 F.3d at 1339 ("[W]e are not faced with a situation where general-purpose computer components are added post-hoc to a fundamental economic practice or mathematical equation.") In contrast to the above, the present claims do not produce as a result an intangible solution to a mathematical equation or solution to a business problem, but the tangible result of an improved distributed computer.

iii. The Machine-or-Transformation Test Also Supports a Finding of Patent-Eligibility as the New Computer Architecture Embodies the Invention, Not Some Underlying Business Method.

The Supreme Court noted that, while not the exclusive test, the machine-or-transformation test is a "useful and important clue" for determining patent-eligibility. *Bilski*, 561 U.S. at 604; *Versata Dev. Grp., Inc. v. SAP Am., Inc.*, 793 F.3d 1306, 1335 (Fed. Cir. 2015). A claimed process is patent-eligible under § 101 if it is tied to a particular machine or apparatus. *Bilski*, 545 F.3d at 954. The machine must play a significant part in permitting the claimed method to be performed, rather than merely achieving a solution more quickly, *i.e.*, through the utilization of a computer for performing calculations. *SiRF Tech., Inc. v. Int'l Trade Comm'n*, 601 F.3d 1319, 1333 (Fed. Cir. 2010). Here, the improvement recited in the claims is not merely using an existing prior art computer to perform a specific calculation more quickly. *See Cf. Versata,* 793 F.3d at 1335; *Bancorp Servs., L.L.C. v. Sun Life Assur. Co. of Canada (U.S.),* 687 F.3d 1266, 1278 (Fed. Cir. 2012). For example, the claims are not directed to applying the alleged abstract idea of "Command and Control" more quickly. Instead, the computer elements recited in the claims are not only essential, they embody the invention – an improved distributed computer. *See, e.g., Research Corp. Techs. v. Microsoft Corp.*, 627 F.3d 859, 869 (Fed. Cir. 2010) ("[I]nventions with specific applications or improvements to technologies in the marketplace are not likely to be so abstract that they override the statutory language and framework of the Patent Act."). Accordingly, patent eligibility is also supported by the claims' satisfaction of the machine-or-transformation test.

iv. The District Court Erred to the Extent It Took Judicial Notice of the *Appistry* I Court's Conclusion That the Patents Are Not Directed to a New Computer.

In making its first prong finding, the District Court stated "[a]s a preliminary matter, Plaintiff simply does not explain how the claims in these patents are any different from those asserted in *Appistry I*." Appx006. Because separate patents describe separate and distinct inventions, the District Court erred to the extent that it relied on another court's prior judgment on related patents. *Comair Rotron, Inc. v. Nippon Densan Corp.*, 49 F.3d 1535, 1539 (Fed. Cir. 1995) ("separate patents describe 'separate and distinct [inventions], 35 U.S.C. § 121; 37 C.F.R. § 1.141, and it can not be presumed that related patents rise and fall together."); *Enzo Biochem, Inc. v. Calgene, Inc.*, 188 F.3d 1362, 1379 (Fed. Cir. 1999) ("Patents, like the claims within them, are independently presumed valid."). To the extent that the District Court was relying on factual findings by the district court in *Appistry I*, the District Court erred because such factual findings are disputed and cannot be taken as true in favor of the Defendants in this case. *U.S. v. Corinthian Colls.*, 655 F.3d 984, 999 (9th Cir. 2011) ("[W]e may not, on the basis of evidence outside of the Complaint, take judicial notice of facts favorable to Defendants that could reasonably be disputed.").

D. The District Court Erred in Holding that the Claims Did Not Contain an Inventive Concept.

Under the second prong of *Alice*, the elements of the claims must be considered, first individually and then as a whole, to determine whether there are additional elements that transform the nature of the claims into a patent-eligible application of the alleged abstract concept. *Alice, 1*34 S. Ct. at 2355. The District Court erred by finding that the claims did not contain an inventive concept because there were unrebutted facts showing that the computer functions claimed were not conventional. The District Court erred in its analysis of the claims as a whole because unrebutted facts showed that the claimed solution solved a problem specific to computers and improved the functioning of the computer itself, just like the inventions recited in *Diehr*, *DDR Holdings*, and *Bascom.*

i. Considered Separately, the Elements Recited in the Claims Are Not Conventional Functions Performed by a Computer.

Under the second step of *Alice,* when examining each element in the claim individually, the Supreme Court has looked to whether the computer functions recited in the elements were "'well-understood, routine, conventional activit[ies]' previously known in the industry." *Id.* at 2359 (quoting *Mayo Collaborative Servs. v. Prometheus Labs., Inc.,* 132 S. Ct. 1289, 1294 (2012)). The District Court does not cite to any evidence that all of the computer functions recited in the elements of the claims were conventional and known in the prior art at the time of the filing of the patents-in-suit. Appx009-010. While the computer functions in the claims may be performed using a conventional computer with the recited novel programming, these functions are not conventional. *See Contentguard Holdings, Inc. v. Amazon.com, Inc.*, No. 2:13-cv-1112, 2015 WL 5853984, at *6 (E.D. Tex. Oct. 5, 2015) ("[T]he Court does not find that all the steps or limitations in the Patents-in-Suit are conventional, even though they may be performed using conventional technology."); *see also Enfish*, 822 F.3d at 1335 ("Software can make non-abstract improvements to computer technology.").

Contrary to the unsupported District Court findings, the specifications and Complaint explain that the computer functions recited in the claims are not conventional or known in the prior art. For example, in the prior art of distributing computing, large problems were broken up into smaller tasks, "spread across many small computers, solved *independently*, and then brought back together." Appx054: Col. 3, Lines 20-24. Importantly, in this prior art, a single independent task of a larger problem, as opposed to a task *sequentially dependent* on other tasks for solving the overall problem, was performed independently by each individual computer. *Id*. Due to the independence of the assigned tasks, in the prior art system there was no need to, and the individual computer did not, compute or distribute state information. Appx054: Col. 3, Lines 20-24; Lines 42-50. Thus, the detailed computer functions recited in the claims here addressing the "processing job," "process flow," "task result," and "state information" were not known in the art. This is supported by the allegations in the Complaint and the Green Declaration. Appx128-129, ¶¶ 57-58; Appx132-140, ¶¶ 89-106; Appx251-260, ¶¶ 34-53. As one example, the prior art computers did not include the following element: "the process handler to which the processing job data was communicated being configured to (1) receive the communicated processing job data, and (2) analyze the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers;" as recited in claim 1 of the '959 Patent. Appx066: Col. 27, Lines 62-67.

Thus, the District Court's unsupported finding that "the claims here do little more than task generic computers with generic functions" must be rejected. Appx009. The claims recite unconventional elements, more than any alleged abstract concept, that confine the claims to a particular, useful application and thus are patent-eligible. *Mayo,* 132 S. Ct. at 1300 ("[T]he claimed process included not only a law of nature but also several unconventional steps . . . that confined the claims to a particular, useful application of the principle."); *Diehr*, 450 U.S. at 187 ("Their process admittedly employs a well-known mathematical equation, but they do not seek to pre-empt the use of that equation. Rather, they seek only to foreclose from others the use of that equation in conjunction with all of the other steps in their claimed process."); *see also Waxham v. Smith*, 294 U.S. 20, 21–22 (1935) ("By the use of materials in a particular manner, he secured the performance of the function by a means which had never occurred in nature and had not been anticipated in the prior art; this is a patentable method or process.").

ii. The Elements of the Claims, Considered as a Whole, Improved upon the Performance of Distributed Computing in a Way That Was Not Previously Known to the Industry.

Even if this Court finds that, individually, every element of each claim recites a well-known computer function at the time before the filing of the Appistry

Patents—a finding unsupported by the record—the inquiry does not end there. The claims must be considered as a whole to determine whether they merely recite an abstract concept as performed by a generic computer. *Alice Corp.*, 134 S. Ct. at 2355. The Supreme Court has stated:

> In determining the eligibility of respondents' claimed process for patent protection under § 101, their claims must be considered as a whole. It is inappropriate to dissect the claims into old and new elements and then to ignore the presence of the old elements in the analysis. This is particularly true in a process claim because a new combination of steps in a process may be patentable even though all the constituents of the combination were well known and in common use before the combination was made.

Diehr, 450 U.S. at 188; *see also Expanded Metal Co. v. Bradford*, 214 U.S. 366, 381 (1909) ("It is perfectly well settled that a new combination of elements, old in themselves, but which produce a new and useful result, entitles the inventor to the protection of a patent"). This Court recently recognized these principles stating "[t]he inventive concept inquiry requires more than recognizing that each claim element, by itself, was known in the art." *Bascom*, 827 F.3d at 1346. Rather, "an inventive concept can be found in the non-conventional and non-generic arrangement of known, conventional pieces." *Id.*

As set forth below, similar to the inventions recited in *Diehr, Bascom* and *DDR Holdings*, considered as a whole, the claims effect an improvement in distributed computing, include additional elements that do not merely recite implementation of an abstract concept by a generic computer, and are, at a minimum, patent-eligible applications of the alleged abstract concept. In *Alice*, the Supreme Court distinguished between patent-ineligible claims that implement an abstract concept on a generic computer and patent-eligible method claims that "for example, purport to improve the functioning of the computer itself." *Alice Corp.*, 134 S. Ct. at 2359. The Supreme Court, citing its decision in *Diehr*, also distinguished patent-ineligible claims from those that "effect an improvement in any other technology or technical field." *Id.* In *Diehr*, the Court upheld as patent-eligible a process for curing rubber that used a computer and a mathematical equation (abstract idea). *Id.* at 2358. The *Diehr* claim used the abstract idea in "a process designed to solve a technological problem in 'conventional industry practice.'" *Id.* at 2358 (quoting *Diehr*, 450 U.S. at 178). The claim included additional steps that recorded temperature measurements inside a rubber mold, something that the industry had not been able to obtain, and the computer used those measurements to repeatedly recalculate the remaining cure time. *Id*. "[This] overall process [was] patent eligible because of the way the additional steps of the process integrated the equation into the process as a whole." *Mayo*, 132 S. Ct. at 1298. "[T]he claims in *Diehr* were patent eligible because they improved an existing technological process, not because they were implemented on a computer." *Alice Corp.*, 134 S. Ct. at 2358.

Considered with the commentary by the Court in *Mayo*, the claims in *Diehr* were patent-eligible because the "steps, or at least the combination of those steps, were [not] in context obvious, already in use, or purely conventional" and resulted in an improved technological process. *Mayo*, 132 S. Ct. at 1299; *Alice Corp.*, 134 S. Ct. at 2358. Just as in *Diehr*, the claims in the Appistry Patents recite a particular configuration not previously used resulting in an overall system that improves the functioning of a technological product—a distributed computer. *See Bascom*, 827 F.3d at 1350–51 ("[T]he claims are more than a drafting effort designed to monopolize the abstract idea. Instead, the claims may be read to improve an existing technological process.") (internal quotations omitted).

In *DDR Holdings, LLC v. Hotels.com, L.P.*, 773 F.3d 1245, 1259 (Fed. Cir. 2014), this Court found computer implemented claims patent-eligible. The claims in that case involved generating a composite web page in response to a user's click on an Internet advertisement. *Id*. at 1248–49. In the prior art, if a website visitor clicked on a merchant's ad, that visitor would be lured from the host website to the merchant's website. *Id*. at 1248. The patent claims solved the problem of host websites losing visitor traffic to ads by creating a new webpage that, when a user clicks on a merchant's ad, directs the website visitor to a composite page that displays product information from the merchant but retains the host website's look and feel. *Id.* at 1248–49. In this way, the host website retains the visitor traffic. *Id.* at 1249.

Applying step two of *Alice*, the Court found the claims patent-eligible because when the limitations of the claims were "taken together as an ordered combination, the claims recite[d] an invention that [was] not merely routine or conventional use of the Internet." *Id.* at 1259. In making this finding, this Court noted that "the claimed solution is necessarily rooted in computer technology in order to overcome a problem specifically arising in the realm of computer networks." *Id.* at 1257. This Court explained how the claims address the problem that was specific to the Internet and why the problem did not arise outside the Internet. *Id.* at 1257–58. The Court noted that, instead of broadly and generically claiming use of the Internet to perform an abstract business practice, the claims specified "how interactions with the Internet are manipulated to yield a desired result- a result that overrides the routine and conventional sequence of events ordinarily triggered by the click of a hyperlink." *Id.* at 1258.

In *Bascom Global Internet Services, Inc.*, the claimed invention provided a system for filtering Internet content (*i.e.*, determined the type of information that could be received over the Internet). *Bascom*, 827 F.3d at 1343. The district court dismissed the plaintiff's claim for failure to state a claim, and this Court vacated and remanded for further proceedings. *Id.* Finding the *Alice* step one determination to be a “close call,” this Court “defer[red] [its] consideration of the specific claim limitations’ narrowing effect for step two.” *Id.* at 1349. In its step two analysis, this Court first concluded that the claims, which “recite[d] a generic computer, network and Internet components,” were not inventive when considered individually. *Id.* However, this Court found an inventive concept in the ordered combination of limitations. *Id.*

In finding that the claims did not merely recite the abstract idea with the requirement to perform it on the Internet, this Court noted that the patent specification explained the problems with existing prior art filters and that while “[f]iltering content on the Internet was already a known concept, [] the patent describes how its particular arrangement of elements is a technical improvement over prior art ways of filtering such content.” *Id.* at *7. Comparing the claims to those in *DDR*, this Court stated “[t]he ‘606 patent is instead claiming a technology-based solution (not an abstract-idea-based solution implemented with generic technical components in a conventional way) to filter content on the Internet that overcomes existing problems with other Internet filtering systems.” Thus, this Court concluded that “the claimed invention represents a ‘software-based invention[] that improve[s] the performance of the computer system itself.’” *Id.*

Similar to *DDR Holdings* and *Bascom*, the specification in the Appistry Patents explains that the present claims address a problem specifically arising in the realm of computers. The specification discusses how prior art distributed computers were deficient with respect to performance, reliability, scalability, and availability. Appx053: Col. 1, Lines 33-34; Col. 1, Lines 44-50; Col. 2, Lines 13-21; Col. 2, Lines 34-41; Col. 2, Lines 43-46; Appx054: Col. 3, Lines 42-50; Col. 4, Lines 1-10; Appx055:Col. 5, Lines 18-22. Facts recited in the specification support a finding that the particular arrangement of computing components claimed in the Appistry Patents was designed to solve technological problems in conventional distributed computers. Appx248-249, ¶¶ 14-21; Appx124-125, ¶¶ 28-35. The specification of the '959 Patent explains that a group of networked hive engines, performing specific functions as recited in the claims, can work together to provide vast computing power. *See, e.g.*, Appx060: Col. 15, Lines 32-42; Appx055: Col. 5, Line 26-Col. 6, Line 7. The result is a system configuration with dramatically improved availability, scalability, performance, and reliability. Appx026: Abstract; Appx055: Col. 5, Lines 18-22; Appx058: Col. 11, Lines 35-48; Col. 12, Lines 14-19; Col. 12, Line 57-67; Appx059: Col. 13, Lines 1-12; Col. 14, Lines 56-62; Appx060: Col. 16, Lines 56-61; Appx061:Col. 17, Lines 21-25; Col. 18, Lines 44-48; Appx063: Col. 21, Lines 37-51; Col. 22, Lines 24-35.

The Complaint also supports a finding that the inventions improve how prior art distributed computers operated stating that the inventions recited in the claims were "a breakthrough technology in high performance computing," "resulted in a successful ongoing business… specializing in high performance computing (HPC) technology utilized in areas such as intelligence, defense, life sciences, financial services, and transportation," and provided "transactional reliability." Appx121-122: ¶¶ 9-10, 12; *see also* Appx124-125, ¶¶ 28-35. Dr. Green's testimony also supports such a finding. Appx248-249, ¶¶ 14-21.

Therefore, just as in *DDR Holdings* and *Bascom,* the claims are patent-eligible because their elements, taken together as an ordered combination, recite a solution that results in an improvement over the prior art distributed computer such that the claimed distributed computer is not operating in its normal, expected manner. *DDR Holdings, LLC,* 773 F.3d at 1258-59 (finding computer-implemented claim patent-eligible where claims recited elements that resulted in computer network that did not operate in its normal, expected manner); *see also Rapid Lit. Mgmt. Ltd. v. CellzDirect, Inc.*, No. 2015-1570, 2016 WL 3606624, at *7 (Fed. Cir. Jul. 5, 2016) ("The individual steps of freezing and thawing were well known, but a process of preserving hepatocytes by repeating those steps was itself far from routine and conventional."). This is not a case where a computer is merely employed for its most basic or routine functions such as performing a business process more quickly or accurately. *See, e.g., Bancorp, 6*87 F.3d at 1279 ("[T]he computer merely permits one to manage a stable value protected life insurance policy more efficiently than one could mentally."); *OIP Techs., Inc. v.*

Amazon.com, Inc., 788 F.3d 1359, 1363 (Fed. Cir. 2015) (finding claims ineligible that "describe the automation of the fundamental economic concept of offer-based price optimization" where the computer was merely relied on to "perform routine tasks more quickly or more accurately.").

iii. The Lack of a Preemption Concern Supports a Finding of Patent-Eligibility.

In *Alice*, the Supreme Court, referring to the abstract idea exception, stated: "[w]e have described the concern that drives this exclusionary principle as one of pre-emption." *Alice Corp.*, 134 S. Ct. at 2354. Even if the alleged abstract concept of Command and Control was somehow included in the claims, there is no concern about preemption of the abstract idea. The claims of the Appistry Patents can only be infringed if the alleged abstract idea of "Command and Control" is used in conjunction with many other specific detailed components and functions recited in the claims. *Mayo*, 132 S. Ct. 1289, 1299 (2012); *DDR Holdings*, 773 F.3d at 1259; *compare O'Reilly v. Morse*, 56 U.S. 62, 112–13 (1853).

The Appistry Patents do not claim every method of improving distributed computers. Instead, as set forth previously, the claims recite a specific improved distributed computer to solve the problems in prior art distributed computers. *See, e.g.*, Appx066: Col. 24, Lines 49-Col. 28, Line 12. Facts identified in the pleadings show that there are other configurations for distributed computing that do not infringe the claims of the Appistry Patents, such as SETI and mainframes.

Appx128. ¶¶ 57-58; Appx132-140, ¶¶ 89-106; Appx251-260, ¶¶ 34, 36-53; Appx054: Col. 3, Lines 18-59. This Court held in *DDR* that this supports a finding of patent-eligibility stating: "It is also clear that the claims at issue do not attempt to preempt every application of the idea . . . Rather, they recite a specific way . . . in order to solve a problem faced by websites on the Internet." *DDR*, 773 F.3d at 1259.

In sum, there is no concern that the claims of the Appistry Patents would preempt future configurations of distributed computing or use of the alleged abstract idea of Command and Control, further supporting patent-eligibility. *Rapid Lit. Mgmt.*, 827 F.3d at 1052 ("[W]hile pre-emption is not the test for determining patent-eligibility, it is certainly the concern that undergirds § 101 jurisprudence.") (internal quotations and citations omitted); *Bascom*, 827 F.3d at 1350 (finding lack of preemption to support a finding of patent-eligibility).

iv. In Light of the Similarities Between Obviousness and § 101 Inquiries, Policy Reasons Support Considering Copying and Commercial Success in the § 101 Inquiry and Support a Finding of Inventive Concept Here.

This Court has recognized that "analysis of § 101 is facilitated by considerations analogous to those of §§ 102 and 103 . . . ." *Internet Patents Corp. v. Active Network, Inc.*, 790 F.3d 1343, 1347 (Fed. Cir. 2015). This Court has stated that "[d]etermination of what is an inventive concept favors inquiries analogous to those undertaken for determination of patentable invention, for a known idea, or one that is routine and conventional, is not inventive in patent terms, as the Court found in *Bilski, Mayo, and Alice*." *Id*. at 1346. Indeed, the § 101 inquiry of whether the function performed by the computer at each step is purely conventional is almost identical to the § 103 inquiry of "whether a patent claiming a combination of known elements would have been obvious, [where] we must ask whether the improvement is more than the predictable use of prior art elements according to their established functions." *TriMed, Inc. v. Stryker Corp.*, 608 F.3d 1333, 1341 (Fed. Cir. 2010) (internal citations and quotations omitted). In this case, the District Court expressly considered obviousness in determining whether the claimed particular configuration was conventional. *See* Appx009 ("A conventional element may be one that is ubiquitous in the field, insignificant or obvious.").

In light of this similarity, inclusion of secondary considerations in the inventive concept prong of *Alice* would provide the same benefit as it does in the § 103 context–guarding against the "subconscious reliance on hindsight." *Mintz v. Dietz & Watson, Inc.*, 679 F.3d 1372, 1378 (Fed. Cir. 2012). As the Federal Circuit has explained:

> These objective criteria thus help turn back the clock and place the claims in the context that led to their invention. Technical advance, like much of human endeavor, often occurs through incremental steps toward greater goals. These marginal advances in retrospect may seem deceptively simple, particularly when retracing the path already blazed by the inventor. For these reasons, this court requires

> consideration of these objective indicia because they provide objective evidence of how the patented device is viewed in the marketplace, by those directly interested in the product.

Id. (internal citations and quotations omitted). For these reasons, secondary considerations of copying and commercial success should be considered and support a finding of an inventive concept in this case.

In this case, the commercial success and industry praise of the inventions recited in the claims supports a finding that the claims recite an inventive concept and are patent-eligible. Appx121-122, ¶¶ 9-10; Appx129-131, ¶¶ 60-78; Appx141, ¶¶ 112-114; Appx142, ¶¶ 115-116; Appx261-262, ¶¶59-63. There is also evidence of copying of the inventions recited in the claims of the Appistry Patents. Factual recitations in the Complaint allege details of how Amazon copied the inventions recited in the claims of the Appistry Patents. Appx122-124, ¶¶ 11-25; Appx141, ¶¶ 117-120. Thus, commercial success, industry praise, and copying of the invention provide additional support for a finding of an inventive concept.

E. The District Court Erred in Resolving Factual Issues at the Motion to Dismiss Stage.

i. Both Prongs of the § 101 Determination Involve Factual Inquiries in This Case.

This Court has repeatedly confirmed that the § 101 inquiry may contain underlying factual issues. *Versata*, 793 F.3d at 1336 ("The section 101 analysis applied by the PTAB was not legally erroneous under *Mayo* and *Alice*. And its underlying fact findings and credibility determinations are supported by substantial evidence in the record."); *Accenture Global Servs., GmbH v. Guidewire Software, Inc.*, 728 F.3d 1336, 1341 (Fed. Cir. 2013); *In re Comiskey*, 554 F.3d 967, 975 (Fed. Cir. 2009).

The Supreme Court has recognized that these underlying factual determinations include "the state of the prior art in the field and the nature of the advancement embodied in the invention." *Microsoft Corp. v. i4i Ltd. P'ship*, 131 S. Ct. 2238, 2242 (2011) ("In evaluating whether [§§ 101, 102, and 103] and other statutory conditions have been met, PTO examiners must make various factual determinations—for instance, the state of the prior art in the field and the nature of the advancement embodied in the invention."). Thus, under the first and second step of *Alice*, when determining whether the claims are directed to an improvement in existing technology, the scope of the invention and how it compares to the state of the prior art can raise factual questions. *See Enfish*, 822 F.3d at 1337 (analyzing scope of invention when determining whether claims were directed to an improvement in technology). Under the second step of *Alice* when examining each element in the claim individually, the inquiry of whether the computer functions recited in the elements were "'well-understood, routine, conventional activit[ies]' previously known in the industry" can also be a factual one. *Alice*, 134 S. Ct. at 2357.

As further support that these inquiries can require the resolution of factual issues, the above issues are nearly identical to the § 103 issues that are factual. For example, the inquiry of "whether the improvement is more than the predictable use of prior art elements according to their established functions" is factual. *TriMed,* 608 F.3d at 1341. Also, the determination of "the character and number of differences between the claimed invention and the prior art" is a fact issue. *Source Search Techs., LLC v. LendingTree, LLC*, 588 F.3d 1063, 1073 (Fed. Cir. 2009). Both the Supreme Court and this Court have recognized this similarity between the factual inquiries present in § 101 and §§102 and 103. *Mayo,* 132 S. Ct. at 1304 ("We recognize that, in evaluating the significance of additional steps, the § 101 patent-eligibility inquiry and, say, the § 102 novelty inquiry might sometimes overlap."); *Internet Patents Corp.,* 790 F.3d at 1346 ("Determination of what is an inventive concept favors inquiries analogous to those undertaken for determination of patentable invention, for a known idea, or one that is routine and conventional, is not inventive in patent terms.").

Thus, whether a computer function is conventional and whether an invention improves the functioning of the computer itself can be factual issues.

ii. Citing Nothing in the Record, and Contrary to the Facts on the Record, the District Court Improperly Resolved Factual Issues by Finding That the Claimed Computer Functions Were Conventional and That the Invention Did Not Improve the Functioning of the Computer Itself.

The District Court found that "the networked computers utilized in the '959 and '267 Patents are merely generic computers tasked with performing generic functions" and "the claims are not directed to solving a technological problem or solve a challenge particular to a specific environment, nor do they contemplate some 'new' type of computer." Appx007-008. Making the above factual findings without any support in the record, the District Court necessarily resolved disputed facts in favor of the Defendant and rejected the facts set out in the specification and Complaint. This was improper since factual allegations in the pleadings are to be accepted as true in ruling on a motion to dismiss. *Chavez v. U.S.*, 683 F.3d 1102, 1108 (9th Cir. 2012); *Lee v. City of Los Angeles*, 250 F.3d 668, 690 (9th Cir. 2001) ("[W]e hold that the district court erred…by taking judicial notice of disputed matters of fact to support its ruling.").

Moreover, the District Court erred by making the above factual findings, without citing any support in the record. *See Lee*, 250 F.3d at 689 ("for purposes of the Rule 12(b)(6) motions to dismiss, there was no basis for the district court to make the factual finding" and district court erred by considering evidence extrinsic to the Complaint); *see also U.S. v. Corinthian Colleges*, 655 F.3d 984, 999 (9th Cir. 2011) ("[W]e may not, on the basis of evidence outside of the Complaint, take judicial notice of facts favorable to Defendants that could reasonably be disputed."). For example, there is no evidence in the record that, at the time of the filing of the patents, the recited logical structure was conventional and that the claims do not improve the functioning of the computer itself. Appx009-011; *TriMed*, 608 F.3d at 1343 ("Merely saying that an invention is a logical, commonsense solution to a known problem does not make it so."); *Plantronics, Inc. v. Aliph, Inc.*, 724 F.3d 1343, 1354 (Fed. Cir. 2013) ("[A] district court's conclusions with respect to obviousness must find support in the record.").

In sum, the District Court erred in granting the motion to dismiss by not only resolving factual issues at the Rule 12 stage, but also by finding these facts without evidentiary support.

iii. Given the Facts on the Record, Including Unrebutted Expert Testimony, the District Court Erred in Finding That the Claims Are Not Patent-Eligible by Clear and Convincing Evidence.

Because each claim of a patent is presumed valid under 35 U.S.C. § 282, Amazon had the burden of proving that the claims are patent-ineligible under § 101 by clear and convincing evidence. *i4i Ltd. P'ship*, 131 S. Ct. at 2242; *CLS Bank*, 717 F.3d at 1284, *aff'd*, 134 S. Ct. 2347 (2014). The clear and convincing standard is a "high bar." *Commil USA, LLC v. Cisco Sys., Inc.*, 135 S. Ct. 1920, 1929 (2015). The District Court did not cite to any support for a factual finding that the computer functions recited in the claims were conventional and that the inventions did not improve the functioning of the computer itself. Appx009-011. Unsupported, conclusory statements are insufficient to meet the clear and convincing standard for a finding of patent-ineligibility. *See Crown Operations Int'l, Ltd. v. Solutia Inc.,* 289 F.3d 1367, 1377–78 (Fed. Cir. 2002) ("Given the presumption of validity afforded the '511 patent, Crown has failed to meet its burden because it has not presented sufficient evidence to rebut the facial evidence offered by Solutia that the Gillery patent does not disclose the two percent limitation. . . . Instead, Crown offers only an assumption and its own contentions."); *Golden Blount, Inc. v. Robert H. Peterson Co.*, 365 F.3d 1054, 1061–62 (Fed. Cir. 2004) ("Peterson offers merely the bare assertion that the patent claims would have been obvious" which failed to meet the clear and convincing standard); *Tech. Lic. Corp. v. Videotek, Inc.*, 545 F.3d 1316, 1339 (Fed. Cir. 2008) (finding trial court did not err in concluding claim valid where party attempting to invalidate the patent failed to introduce evidence as to what prior art "was actually available").

Further, the clear and convincing standard cannot be satisfied in this case because the Complaint and the specifications provide *unrebutted* facts supporting an inventive concept. *See, e.g., S3 Inc. v. NVIDIA Corp.*, 259 F.3d 1364, 1371 (Fed. Cir. 2001) ("There was no contrary evidence. . . . Thus the ruling in invalidity for failure to comply with § 112 is incorrect, and must be reversed."); *Atmel Corp. v. Info. Storage Devices, Inc.*, 198 F.3d 1374, 1382 (Fed. Cir. 1999) ("The record indicates that that testimony was essentially unrebutted. That being the case, we conclude that summary judgment was improperly granted invalidating the ‘811 patent . . . .”). This conclusion is further supported by the fact that the ‘267 Patent issued after *Alice* and after the USPTO issued its guidelines to patent examiners for analyzing patents under *Alice*. Appx131-132, ¶¶ 80-88.

iv. The Grant of the Motion to Dismiss Based on “Facts” Determined by the District Court Violated Appistry’s Fifth Amendment Right to Due Process.

Just like disputed issues of fact under § 103, the disputed issues of fact in the present case should be decided by a jury. *Connell v. Sears, Roebuck & Co.*, 722 F.2d 1542, 1547 (Fed. Cir. 1983) (“[O]bviousness is reached after answers to a series of potential fact questions have been found . . . . In the ordinary patent case, the trier of fact must answer the factual inquiries outlined in *Graham v. John Deere Co*.”); *Innogenetics, N.V. v. Abbott Lbs.*, 512 F.3d 1363, 1378 n.6 (Fed. Cir. 2008) (“[D]isputes over material facts . . . should be resolved at trial by the fact finder.”); *see also Retractable Techs., Inc. v. Becton, Dickinson and Co*., 653 F.3d 1296, 1310–11 (Fed. Cir. 2011); *Cushman v. Shinseki*, 576 F.3d 1290, 1296–1300 (Fed. Cir. 2009) (finding violation of due process right to a fair hearing on factual issues of claim). Thus, the District Court erred in granting the motion to dismiss and denying Appistry due process and its right to a jury trial on disputed factual issues.

CONCLUSION

For the foregoing reasons, this Court should reverse the grant of Amazon's Motion to Dismiss.

DATED: October 4, 2016

By: */s/ Anthony G. Simon*

Anthony G. Simon
Timothy D. Krieger
Benjamin R. Askew
Michael P. Kella
THE SIMON LAW FIRM, P.C.
800 Market Street, Suite 1700
Saint Louis, Missouri 63101
P. 314.241.2929
F. 314.241.2029

Attorneys for Plaintiff-Appellant
Appistry, LLC

ADDENDUM

HONORABLE RICHARD A. JONES

UNITED STATES DISTRICT COURT
WESTERN DISTRICT OF WASHINGTON
AT SEATTLE

APPISTRY, INC., Plaintiff, v. AMAZON.COM, INC., et al., Defendants.	CASE NO. C15-1416RAJ ORDER

I. INTRODUCTION

This matter comes before the court on Defendants Amazon.com, Inc. and Amazon Web Services, Inc.'s (collectively "Amazon" or "Defendants") Motion to Dismiss for Invalidity Under 35 U.S.C. § 101 on the grounds that the two patents asserted by Plaintiff Appistry, Inc.[1] ("Appistry" or "Plaintiff") cover ineligible subject matter. *See* Dkt. # 36. Having considered the Parties' arguments, the Court hereby **GRANTS** Amazon's Motion.

II. BACKGROUND

This case concerns U.S. Patent Nos. 8,682,959 and 9,049,267 (the "'959 Patent" and "'267 Patent," respectively). The '959 Patent is entitled "System and Method for Fault Tolerant Processing of Information Via Networked Computers Including Request Handlers, Process Handlers, and Task Handlers." *See* Compl. Ex. 1 ('959 Patent). The '267 Patent is entitled "System and Method for Processing Information Via Networked

[1] While this Motion was pending, Appistry, LLC substituted as Plaintiff. *See* Dkt. # 47.

Computers Including Request Handlers, Process Handlers, and Task Handlers." *See id.* Ex. 2 ('267 Patent). Both patents are child patents of U.S. Patent Nos. 8,200,746 and 8,341,209 (the "'746 Patent" and "'209 Patent," respectively). *See* '959 Patent at 1; '267 Patent at 1. The '746 and '209 Patents have since been held to be invalid under 35 U.S.C. § 101. *See Appistry, Inc. v. Amazon.com, Inc.* ("*Appistry I*"), No. C15-311 MJP, 2015 WL 4210890, at *5 (W.D. Wash. July 9, 2015).

The '959 and '267 Patents have the same inventors, figures, and "Detailed Descriptions" as the '746 and '209 Patents. *Compare* '959 Patent & '267 Patent *with* Case No. C15-311MJP, Dkt. # 1-1 ('746 Patent) & Dkt. # 1-2 ('209 Patent). Generally, all four patents relate to using "[a] hive of computing machines . . . to process information." *See* '959 Patent at 8:32-33. To do so, the claimed inventions use a system of "a plurality of networked computers" to "process[] a plurality of processing jobs in a distributed manner." *See* '959 Patent at 31:30-31; '267 Patent at 28:8-9. To do so, the claimed systems enlist "a request handler, a plurality of process handlers, and a plurality of task handlers." *See* '959 Patent at 31:32-34; '267 Patent at 28:10-12.

III. LEGAL STANDARD

Fed. R. Civ. P. 12(b)(6) permits a court to dismiss a complaint for failure to state a claim. The rule requires the court to assume the truth of the complaint's factual allegations and credit all reasonable inferences arising from those allegations. *Sanders v. Brown*, 504 F.3d 903, 910 (9th Cir. 2007). A court "need not accept as true conclusory allegations that are contradicted by documents referred to in the complaint." *Manzarek v. St. Paul Fire & Marine Ins. Co.*, 519 F.3d 1025, 1031 (9th Cir. 2008). The plaintiff must point to factual allegations that "state a claim to relief that is plausible on its face." *Bell Atl. Corp. v. Twombly*, 550 U.S. 544, 568 (2007). If the plaintiff succeeds, the complaint avoids dismissal if there is "any set of facts consistent with the allegations in the complaint" that would entitle the plaintiff to relief. *Id.* at 563; *Ashcroft v. Iqbal*, 556 U.S. 662, 679 (2009).

A court typically cannot consider evidence beyond the four corners of the complaint, although it may rely on a document to which the complaint refers if the document is central to the party's claims and its authenticity is not in question. *Marder v. Lopez*, 450 F.3d 445, 448 (9th Cir. 2006). A court may also consider evidence subject to judicial notice. *United States v. Ritchie*, 342 F.3d 903, 908 (9th Cir. 2003).

IV. ANALYSIS

Before the Court proceeds to whether the '267 and 959 Patents are directed to patent-eligible subject matter, it is necessary to determine whether claim 29 of the '959 Patent and claim 1 of the '267 Patent are representative. Plaintiff does not agree that the claims are representative. *See* Dkt. # 38 at 12. Citing a few minute details between these claims and those at issue in *Appistry I*, Plaintiff argues that *all* claims in the '959 and '267 Patents must be independently reviewed. But Plaintiff's own comparison of the independent claims in the '959 and '267 Patents reveals just how similar (and representative) claim 29 and claim 1 are. *See* Dkt. # 39-1. For example, the claims use practically identical language – including, crucially, the "request handlers," "process handlers," and "task handlers" utilized to distribute work. The Court finds that the claims are representative[2] and will proceed on that basis.[3]

Section 101 of the Patent Act defines patent-eligible subject matter, providing that "[w]hoever invents or discovers any new and useful process, machine, manufacture, or

[2] Certain other independent claims do contain other claim elements. For example, claims 1, 11, 27, and 28 of the '959 Patent add the limitation that if a fault exists, the process handler or request handler will initiate a recovery procedure. *See* '959 Patent at 28:6-11, 29:10-16, 30:61-67, 31:21-27. Similarly, claim 123 of the '267 Patent incorporates the limitation that process handlers "are configured to volunteer for servicing the processing jobs based on their availabilities." *See* '267 Patent at 39:58-60.

The Court finds that these differences do not make claim 29 of the '959 Patent or claim 1 of the '267 Patent any less representative of the other claims. Ultimately, all of the independent claims cover systems using the different "handlers" to process information and tasks.

[3] The Federal Circuit has already addressed this issue, and held that a court need not address every asserted claim if the claims of the asserted patents "are substantially similar in that they address little more than the same abstract idea" and the selected claims are representative. *See Content Extraction & Transmission LLC v. Wells Fargo Bank, Nat'l Ass'n*, 776 F.3d 1343, 1348 (Fed. Cir. 2014).

composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title." 35 U.S.C. § 101. The Supreme Court has, however, recognized that laws of nature, natural phenomena, and abstract ideas are not patentable. *Alice Corp. v. CLS Bank Int'l*, 134 S. Ct. 2347, 2354 (2014) (quoting *Ass'n for Molecular Pathology v. Myriad Genetics, Inc.*, 133 S. Ct. 2107, 2116 (2013)). The purpose of these exceptions is to protect the "basic tools of scientific and technological work." *Mayo Collaborative Servs. v. Prometheus Labs., Inc.*, 132 S. Ct. 1289, 1293 (2012). However, courts must "tread carefully in construing this exclusionary principle lest it swallow all of patent law." *Alice*, 134 S. Ct. at 2354. In "distinguishing patents that claim laws of nature, natural phenomena, and abstract ideas from those that claim patent-eligible applications of those concepts," courts apply a two-part test. *Alice*, 134 S. Ct. at 2355. Courts must first "determine whether the claims at issue are directed to one of those patent-ineligible concepts." *Id.* If so, then courts must examine "[w]hat else is there in the claims before [them]" by considering "the elements of each claim both individually and 'as an ordered combination' to determine whether the additional elements 'transform the nature of the claim' into a patent-eligible application." *Id.* The Court has characterized this search as one "'inventive concept' — *i.e.*, an element or combination of elements that is 'sufficient to ensure that the patent in practice amounts to significantly more than a patent upon the [ineligible concept] itself.'" *Id.* (quoting *Mayo*, 132 S. Ct. at 1294).

The first step in the Court's analysis is to "determine whether the claims at issue are directed to one of those patent-ineligible concepts." *Alice*, 134 S. Ct. at 2355. Some courts have characterized the first step as distilling "the gist of the claim." *See Open Text S.A. v. Box, Inc.*, 78 F. Supp. 3d 1043, 1046 (N.D. Cal. 2015) (citing cases). The *Appistry I* court confronted claims virtually identical to those asserted here and found that they were directed to "the abstract idea of distributed processing akin to the military's command and control system." *See* 2015 WL 4210890 at *2. The Court finds that the claims in the '959 and '267 Patents are also directed to that abstract idea.

As a preliminary matter, Plaintiff simply does not explain how the claims in these patents are any different from those asserted in *Appistry I*. That is not surprising. As discussed, *supra*, the patents share the same terminology and central idea: that of distributing tasks through a hierarchical structure.

To be sure, there are a few differences between the claims in the '959 and '267 Patents and those in the '209 and '746 Patents. However, those differences are minor. For example, the '959 and '267 Patents require that the "process handlers" and "task handlers" be resident on "different networked computers." *See* '959 Patent at 31:34-37; '267 Patent at 28:12-15 ("the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers."). And the '209 and '746 Patents require no such thing. *See* '746 Patent at 26:53-61 ("wherein a plurality of the hive engines within the at least one territory are configured to perform the processing job in a distributed manner such that the processing tasks of the processing job are distributed to a plurality of hive engines within the at least one territory for execution thereby").

The '959 and '267 Patents also differ in that they include the claim element "the processing jobs having a plurality of associated process flows, the process flows including . . . (2) logic configured to define a relationship between the processing tasks of the same process flow." *See* '959 Patent at 31:37-41; '267 Patent at 28:15-19. But that element appears to simply provide for a sequence of delegating tasks. *See* '959 Patent at 24:26-28 ("One embodiment uses a logical hierarchy of hive engines for delegation of performing administrative and/or other hive related tasks").

Finally, as to the claim elements calling for receiving, storage, or communicating data, those are "undisputably well-known" and do not distinguish the claims of the '959 and '267 Patents from those found to be directed to abstract ideas. *See Content*

Extraction, 776 F.3d at 1347 (Fed. Cir. 2014); *buySAFE, Inc. v. Google, Inc.*, 765 F.3d 1350, 1355 (Fed. Cir. 2014) ("That a computer receives and sends the information over a network—with no further specification—is not even arguably inventive."); *see also Encyclopaedia Britannica, Inc. v. Dickstein Shapiro LLP*, 128 F. Supp. 3d 103, 112-13 (D.D.C. 2015) (collecting cases and holding that "[t]he abstract concept of collecting, storing, and retrieving data simply is not patent-eligible").

Still, Plaintiff contends that the claims here are not directed to abstract ideas, but instead to a new computer (or, more specifically, a more efficiently and reliably distributed configuration of multiple computers), resulting in better performance. *See* Dkt. # 38 at 15. There is at least some basis for this argument. The Federal Circuit held that where a "claimed solution is necessarily rooted in computer technology in order to overcome a problem specifically arising in the realm of computer networks," it is not necessarily directed to an abstract idea. *See DDR Holdings, LLC v. Hotels.com, L.P.*, 773 F.3d 1245, 1257 (Fed. Cir. 2014).

But the Court disagrees with Plaintiff's application here. Plaintiff relies heavily on the specification of the '959 Patent to show that the patents are directed to such a new computer. *See id.* at 16. That same portion of the specification was first presented – verbatim – in the '746 and '209 Patents, which were of course invalidated in *Appistry I*. *See* '746 Patent at 5:11-15; '209 Patent at 5:15-19.

Even beyond that, however, it is clear that the networked computers utilized in the '959 and '267 Patents are merely generic computers[4] tasked with performing generic functions. The specification itself clarifies that "[t]he term 'computer' is used generically

[4] The Court also disagrees with Plaintiff's contention that the machine or transformation test is helpful in this case. *See* Dkt. # 38 at 23-24. "[T]hat test may be a useful and important clue or investigative clue" in determining "whether an invention is patent-eligible." *See Bilski*, 561 U.S. at 594. However, even in the context of this test, "after *Alice,* there can remain no doubt: recitation of generic computer limitations does not make an otherwise ineligible claim patent-eligible." *DDR Holdings*, 773 F.3d at 1256 (citing *Alice*, 134 S. Ct. at 2358). "The bare fact that a computer exists in the physical rather than purely conceptual realm 'is beside the point.'" *Id.* Similarly, here, the fact that the inventions here necessarily apply to generic computers is irrelevant because there simply is no inventive concept embodied within them.

herein to describe any number of computers, including, but not limited to personal computers, embedded processing elements and systems . . . ." *See* '959 Patent at 8:47-51; '267 Patent at 9:1-5. Nor are the claimed systems or methods limited to a particular type of process or task – the patents specify that "[t]he terms 'task' and process are used generically herein to describe any type of running program, including, but not limited to a computer process, task, thread, executing application, operating system, user process, device driver, native code, machine or other language, etc. . . ." *See* '959 Patent at 8:57-66; '267 Patent at 9:11-20. In other words, the claims are not directed to solving a technological problem or solve a challenge particular to a specific environment, nor do they contemplate some "new" type of computer.

The Court also rejects Plaintiff's argument that the claims here are not directed to abstract ideas because they do not claim solutions to mathematical equations or business problems. *See* Dkt. # 38 at 17-18. The *Appistry I* court already rejected this argument for reasons this Court also finds convincing. Simply put, "the operative question is whether or not the patent claims are directed toward an abstract idea, and *not* whether or not the invention could be classified into one of Plaintiff's three categories." *See Appistry I*, 2015 WL 4210890 at *2 (citing *Alice*, 134 S. Ct. at 2356-57).

Plaintiff also contends that the claims here are not directed to abstract ideas because they cannot be performed by humans. *See* Dkt. # 38 at 17-18. But the fact that inventions here are implemented on computers or only exist in the computing realm does not save them. *See Bancorp Servs., L.L.C. v .Sun Life Assurance Co. of Canada (U.S.)*, 687 F.3d 1266, 1279 (Fed. Cir. 2012) (citing *SiRF Tech., Inc. v. Int'l Trade Comm'n*, 601 F.3d 1319, 1332 (Fed. Cir. 2010)) ("Bancorp seeks to analogize its case to *SiRF,* contending that a computer 'plays a significant part' in its claims because they require 'precise and repetitive calculation.' . . . That misses the point."); *see also Cyberfone Sys., LLC v. CNN Interactive Grp., Inc.*, 558 F. App'x 988, 992 (Fed. Cir. 2014) (citing *Bilski* *v. Kappos*, 561 U.S. 593, 610 (2010)) ("the category of patent-ineligible abstract ideas is not limited to methods that can be performed in the human mind.").

Having determined that the '959 and '267 Patents' claims are "directed to" an abstract idea, the Court must next determine if its claims are nevertheless patentable because they contain an "inventive concept" sufficient to "transform the claimed abstract idea into a patent-eligible application." *Alice*, 134 S. Ct. at 2357. To do so, courts are instructed to consider the elements of the claims – both individually and in an ordered combination – to assess whether the elements transform the nature of the claims into a patent-eligible inventive concept. *See Content Extraction*, 776 F.3d at 1347.

The Court must disregard "'well-understood, routine, conventional activit[ies]' previously known to the industry" at this step of the analysis. *See Alice*, 134 S. Ct. at 2359 (quoting *Mayo*, 132 S. Ct. at 1299) (alterations in original). "A conventional element may be one that is ubiquitous in the field, insignificant or obvious." *Cal. Inst. of Tech. v. Hughes Commc'ns Inc.*, 59 F. Supp. 3d 974, 992 (C.D. Cal. 2014) (citing *Mayo*, 132 S. Ct. at 1298). Such a "conventional element may also be a necessary step, which a person or device must perform in order to implement the abstract idea." *Id.* Although "conventional elements and prior art may overlap," "conventional elements do not constitute everything in prior art." *Id.*

The Court finds that whether viewed individually or as an ordered combination, the claims of the '959 and '267 Patents do not contain an inventive concept.

As discussed, *supra*, the claims here do little more than task generic computers with generic functions. Even the central idea of the claims – the use of a "network" of computers – uses a generic network. *See* '959 Patent at 9:12-20 ("the terms 'network' and 'communications mechanism' are used generically herein to describe one or more networks, communications mediums or communications systems, including, but not limited to the Internet, private or public telephone, cellular, wireless, satellite, cable, local area, metropolitan area and/or wide area networks . . . etc."); '267 Patent at 9:33-41.

Simply put, the claims here simply provide for completing a task or process by distributing it downward via a hierarchical series of "handlers" located on generic computers spread throughout a generic network. There is nothing inventive about that.

But Plaintiff holds fast to its contention that as an ordered whole, the claims in the '959 and '267 Patents improve computer function by allowing increased "availability, scalability, performance, and reliability." *See* Dkt. # 38 at 23. That may be possible, but simply limiting this abstract idea of distributed processing to the networked computer field does not alter the analysis or render the idea any more inventive than before. *See buySAFE, Inc.*, 765 F.3d at 1354 (quoting *Alice*, 134 S. Ct. at 2358) ("Neither 'attempting to limit the use of [the idea] to a particular technological environment' nor a 'wholly generic computer implementation' is sufficient" to add an inventive concept").

In fact, the *Appistry I* court addressed the same argument – that the inventions were technological improvements to distributed computing because they permitted improved speed, efficiency, and reliability – and rejected it. *See* 2015 WL 4210890 at *4. The court found that "the actual systems and methods claimed-through which efficiency and reliability are achieved-are well understood, routine, and purely conventional, and do not supply an inventive concept separate from the underlying abstract idea." *Id.* That analysis remains true for the '959 and '267 Patents.

The system claimed by the '267 Patent, for example, uses several networked computers separated into tiered "handlers," including a "request handler," "process handlers," and "task handlers." *See* '267 Patent at 28:7-19. The processing jobs handled by this system have discrete tasks, as well as a sequence for processing those tasks. *See id.* at 28:15-19. The request handler is instructed to receive a job request, store information about that job, and then to communicate it to process handlers. *See id.* at 28:20-24. The process handlers are instructed to analyze information to determine whether any tasks remain to be done and the task handlers are programmed to perform processing tasks. *See* '267 Patent at 28:25-41.

None of this is inventive. Receiving, storing, and sending task data are, of course, "'well-understood, routine, conventional activit[ies]' previously known to the industry." *See OIP Techs., Inc. v. Amazon.com, Inc.*, 788 F.3d 1359, 1363 (Fed. Cir. 2015) (quoting *Alice*, 134 S. Ct. at 2359)); *see also buySAFE, Inc.*, 765 F.3d at 1355 ("That a computer receives and sends the information over a network—with no further specification—is not even arguably inventive."). The other tasks delegated to the process handlers and task handlers do little more than simply check if any tasks remain to be done (*see* '267 Patent at 28:25-39) or simply (and without elaboration) do the assigned task (*see* '267 Patent at 28:40-41). *See Hewlett Packard Co. v. ServiceNow, Inc.*, No. 14-CV-00570-BLF, 2015 WL 1133244, at *10 (N.D. Cal. Mar. 10, 2015) (finding that claim limitation that instructed that "merely instructs that the workflow be verified" did not add an inventive concept). These are not inventive and do little more than simply state the abstract idea – distributed processing through a hierarchical, military-like command structure – and add the words "apply it with a computer." *See Alice*, 134 S. Ct. at 2358.

Finally, the Court addresses several procedural issues that Plaintiff raises.[5]

First, as Plaintiff notes, its Complaint includes the declaration of Dr. Matthew Green (or other allegations regarding the same topics), who opines that the claims of the '959 and '267 Patents cover "a new and novel mechanism, system, and/or method over the prior art for distributed computing." *See* Dkt. # 1-3 (Green Decl.) ¶¶ 13-16. Plaintiff argues that the Court must accept these conclusions as true. *See* Dkt. # 38 at 19-20. That is nonsense. *See Iqbal*, 556 U.S. at 678-79 ("the tenet that a court must accept as true all of the allegations contained in a complaint is inapplicable to legal conclusions"); *Adams v. Johnson*, 355 F.3d 1179, 1183 (9th Cir. 2004) (citing *Sprewell v. Golden State*

[5] Plaintiff also argues that the inventions do not preempt the entire abstract idea. *See* Dkt. # 38 at 24-25. Even if that position is true, it is irrelevant. *See Ariosa Diagnostics, Inc. v. Sequenom, Inc.*, 788 F.3d 1371, 1379 (Fed. Cir. 2015) ("While preemption may signal patent ineligible subject matter, the absence of complete preemption does not demonstrate patent eligibility.").

Warriors, 266 F.3d 979, 988 (9th Cir. 2001)) ("conclusory allegations of law and unwarranted inferences are insufficient to defeat a motion to dismiss").[6]

Second, Plaintiff argues that a "clear and convincing" standard applies to the § 101 inquiry. *See* Dkt. # 38 at 8-9. That is not necessarily incorrect. *See Netflix, Inc. v. Rovi Corp.*, 114 F. Supp. 3d 927, 938 (N.D. Cal. 2015) (citing *Pfizer, Inc. v. Apotex, Inc.*, 480 F.3d 1348, 1359 (Fed. Cir. 2007)). And many courts have recognized a split in authority on this question. *See Papst Licensing GmbH & Co. KG v. Xilinx Inc.*, No. 16-CV-00925-LHK, 2016 WL 3196657, at *7 (N.D. Cal. June 9, 2016) (collecting cases). But in this Court's view, the standard is irrelevant as the outcome would be the same under any standard. Under either standard, Defendants have clearly shown that the claims of the '959 and '267 Patents are directed to an abstract idea.

///

///

///

[6] Plaintiff's position is absurd. Requiring the Court to accept such facts or legal conclusions (even in the form of an early expert declaration) would permit any plaintiff to circumvent the § 101 inquiry on an early motion to dismiss or motion for judgment on the pleadings simply by including a few lines attesting to the novelty of the invention.

US008682959B2

(12) **United States Patent**
Hinni et al.

(10) **Patent No.: US 8,682,959 B2**
(45) **Date of Patent: Mar. 25, 2014**

(54) SYSTEM AND METHOD FOR FAULT TOLERANT PROCESSING OF INFORMATION VIA NETWORKED COMPUTERS INCLUDING REQUEST HANDLERS, PROCESS HANDLERS, AND TASK HANDLERS

(71) Applicant: **Appistry, Inc.**, St. Louis, MO (US)

(72) Inventors: **Aaron Louis Hinni**, Ballwin, MO (US); **Guerry Anderson Semones**, Marthasville, MO (US); **Michael Scott Groner**, Chesterfield, MO (US); **Roberto Raul Lozano**, Creve Coeur, MO (US)

(73) Assignee: **Appistry, Inc.**, St. Louis, MO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/707,861**

(22) Filed: **Dec. 7, 2012**

(65) **Prior Publication Data**

US 2013/0144933 A1 Jun. 6, 2013

Related U.S. Application Data

(60) Continuation of application No. 13/491,893, filed on Jun. 8, 2012, now Pat. No. 8,341,209, which is a continuation of application No. 13/293,527, filed on Nov. 10, 2011, now Pat. No. 8,200,746, which is a division of application No. 12/127,070, filed on May 27, 2008, now Pat. No. 8,060,552, which is a division of application No. 10/236,784, filed on Sep. 7, 2002, now Pat. No. 7,379,959.

(51) **Int. Cl.**
G06F 15/16 (2006.01)
H04L 29/08 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 67/16*** (2013.01)
USPC **709/202**; 709/201; 709/203; 709/223; 709/224

(58) **Field of Classification Search**
CPC H04L 67/16
USPC 709/201–203, 223, 224
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,128,277	A	10/2000	Bruck et al.	
6,665,701	B1 *	12/2003	Combs et al.	718/104
6,766,348	B1 *	7/2004	Combs et al.	718/104
7,035,933	B2	4/2006	O'Neal et al.	
7,043,225	B1 *	5/2006	Patel et al.	455/405
7,379,959	B2	5/2008	Hinni et al.	

(Continued)

OTHER PUBLICATIONS

Beranek et al., "Host Access Protocol Specification", RFC 907, Jul. 1984, 79 pages.

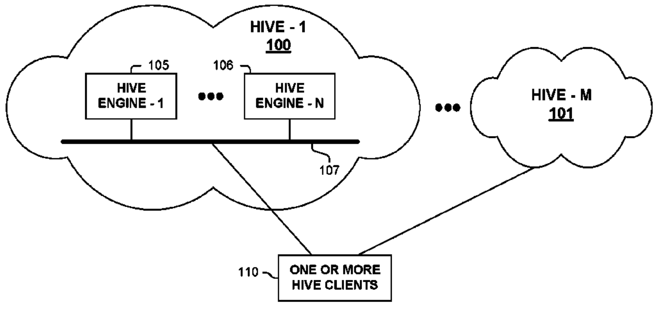
(Continued)

Primary Examiner — Mohamed Wasel
(74) *Attorney, Agent, or Firm* — Thompson Coburn LLP

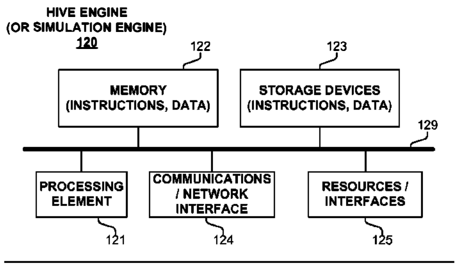
(57) **ABSTRACT**

Systems and methods for processing information via networked computers leverage request handlers, process handlers, and task handlers to provide efficient distributed and fault-tolerant processing of processing jobs. A request handler can receive service requests for processing jobs, process handlers can identify tasks to be performed in connection with the processing jobs, and task handlers can perform the identified tasks, where the request handler, the process handlers, and the task handlers can be distributed across a plurality of networked computers.

48 Claims, 25 Drawing Sheets



EXEMPLARY HIVE COMPUTING ARCHITECTURE


HIVE ENGINE (OR SIMULATION ENGINE)
120
122
123
MEMORY (INSTRUCTIONS, DATA)
STORAGE DEVICES (INSTRUCTIONS, DATA)
129
PROCESSING ELEMENT
COMMUNICATIONS / NETWORK INTERFACE
RESOURCES / INTERFACES
121
124
125

(56) **References Cited**

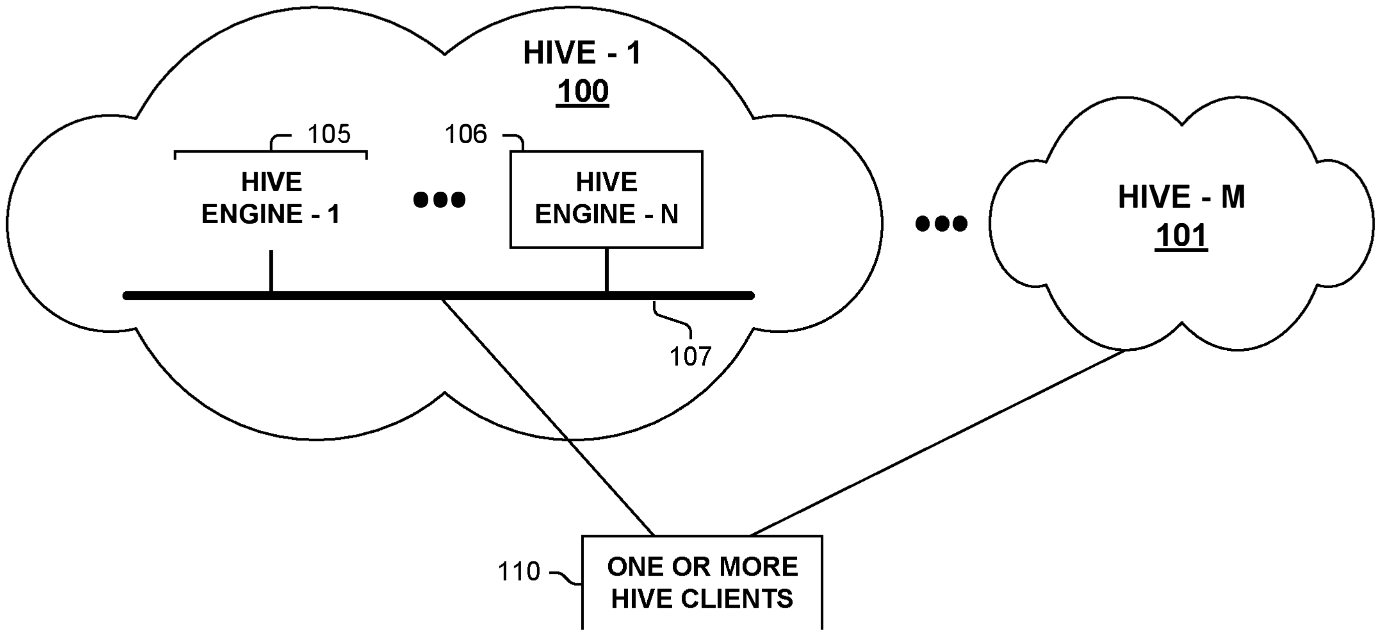
U.S. PATENT DOCUMENTS

8,060,552	B2	11/2011	Hinni et al.	
8,200,746	B2	6/2012	Hinni et al.	
8,341,209	B2	12/2012	Hinni et al.	
2002/0023117	A1	2/2002	Bernardin et al.	
2002/0152106	A1 *	10/2002	Stoxen et al.	705/8
2006/0117212	A1	6/2006	Meyer et al.	
2006/0198386	A1	9/2006	Liu et al.	

OTHER PUBLICATIONS

Coulouris et al., "Distributed Systems Concepts and Design", 2001, pp. 515-552, Third Edition, Chapter 13, Pearson Education Limited.
Coulouris et al., "Distributed Systems Concepts and Design", 2001, pp. 553-606, Third Edition, Chapter 14, Pearson Education Limited.
Postel, "Assigned Numbers", RFC 755, May 3, 1979, 12 pages.
Veizades et al., "Service Location Protocol", RFC 2165, Jun. 1997, 72 pages.

* cited by examiner



EXEMPLARY HIVE
COMPUTING ARCHITECTURE

FIGURE 1A

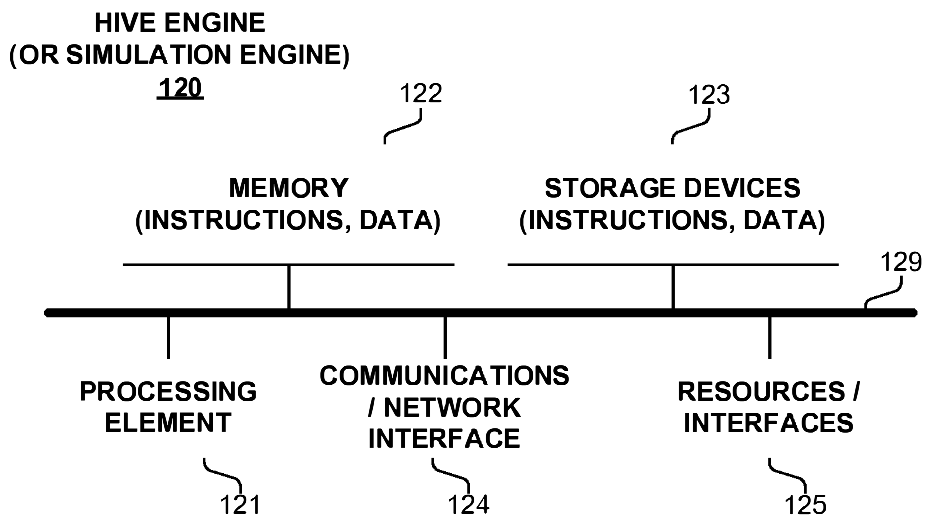

HIVE ENGINE
(OR SIMULATION ENGINE)
120
122
123
MEMORY
(INSTRUCTIONS, DATA)
STORAGE DEVICES
(INSTRUCTIONS, DATA)
129
PROCESSING
ELEMENT
COMMUNICATIONS
/ NETWORK
INTERFACE
RESOURCES /
INTERFACES
121
124
125

FIGURE 1B

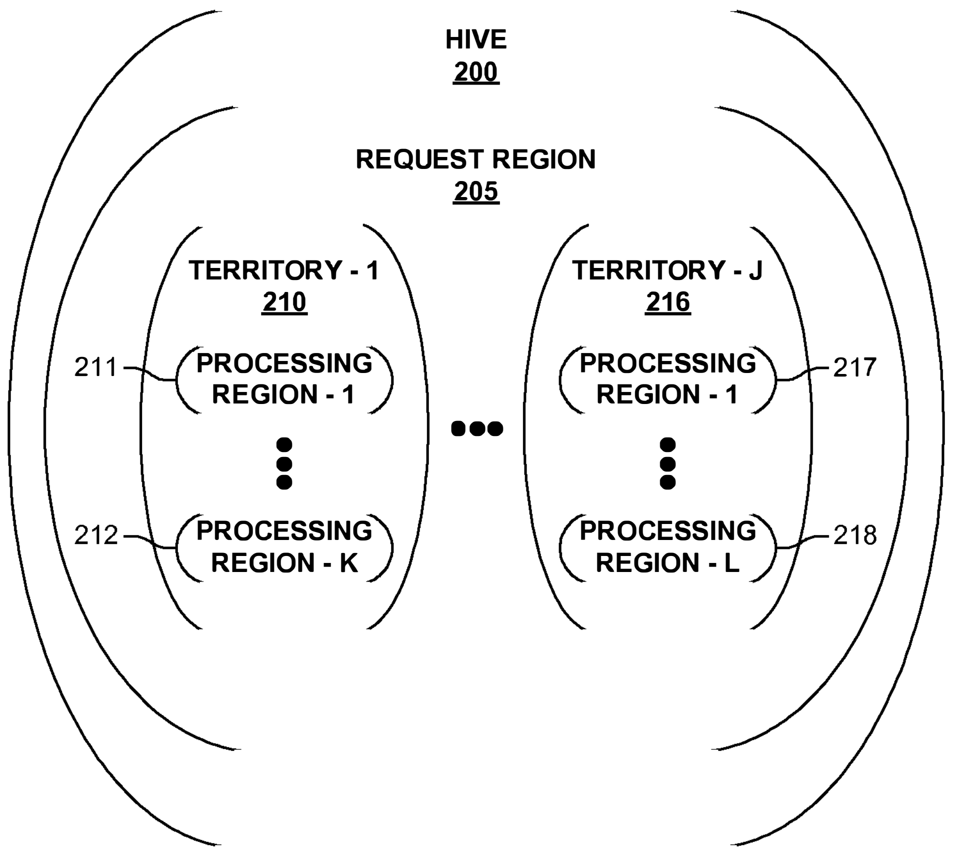

HIVE
200
REQUEST REGION
205
TERRITORY - 1
210
TERRITORY - J
216
211
PROCESSING REGION - 1
212
PROCESSING REGION - K
217
PROCESSING REGION - 1
218
PROCESSING REGION - L

FIGURE 2A

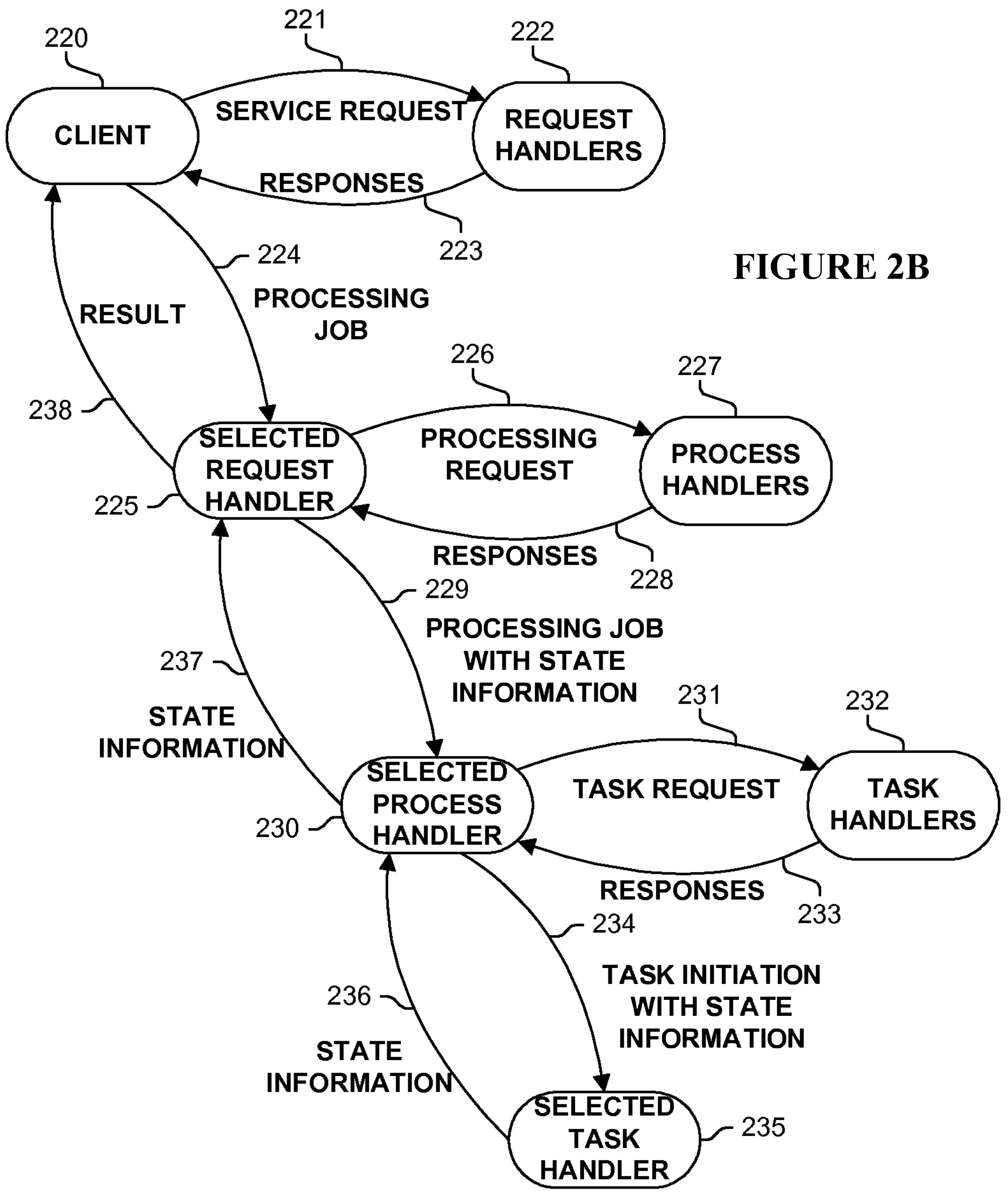

220
CLIENT
221
SERVICE REQUEST
222
REQUEST HANDLERS
RESPONSES
223
224
PROCESSING JOB
RESULT
238
226
PROCESSING REQUEST
227
PROCESS HANDLERS
SELECTED REQUEST HANDLER
225
RESPONSES
228
229
PROCESSING JOB WITH STATE INFORMATION
237
STATE INFORMATION
231
TASK REQUEST
232
TASK HANDLERS
SELECTED PROCESS HANDLER
230
RESPONSES
233
234
TASK INITIATION WITH STATE INFORMATION
236
STATE INFORMATION
SELECTED TASK HANDLER
235

FIGURE 2B

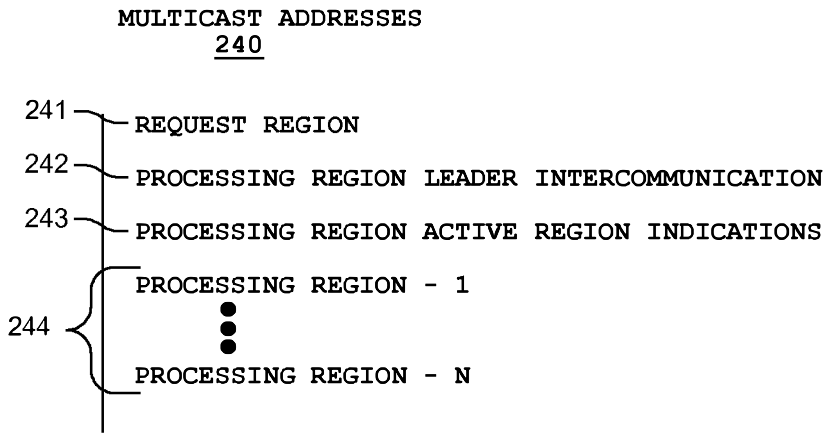
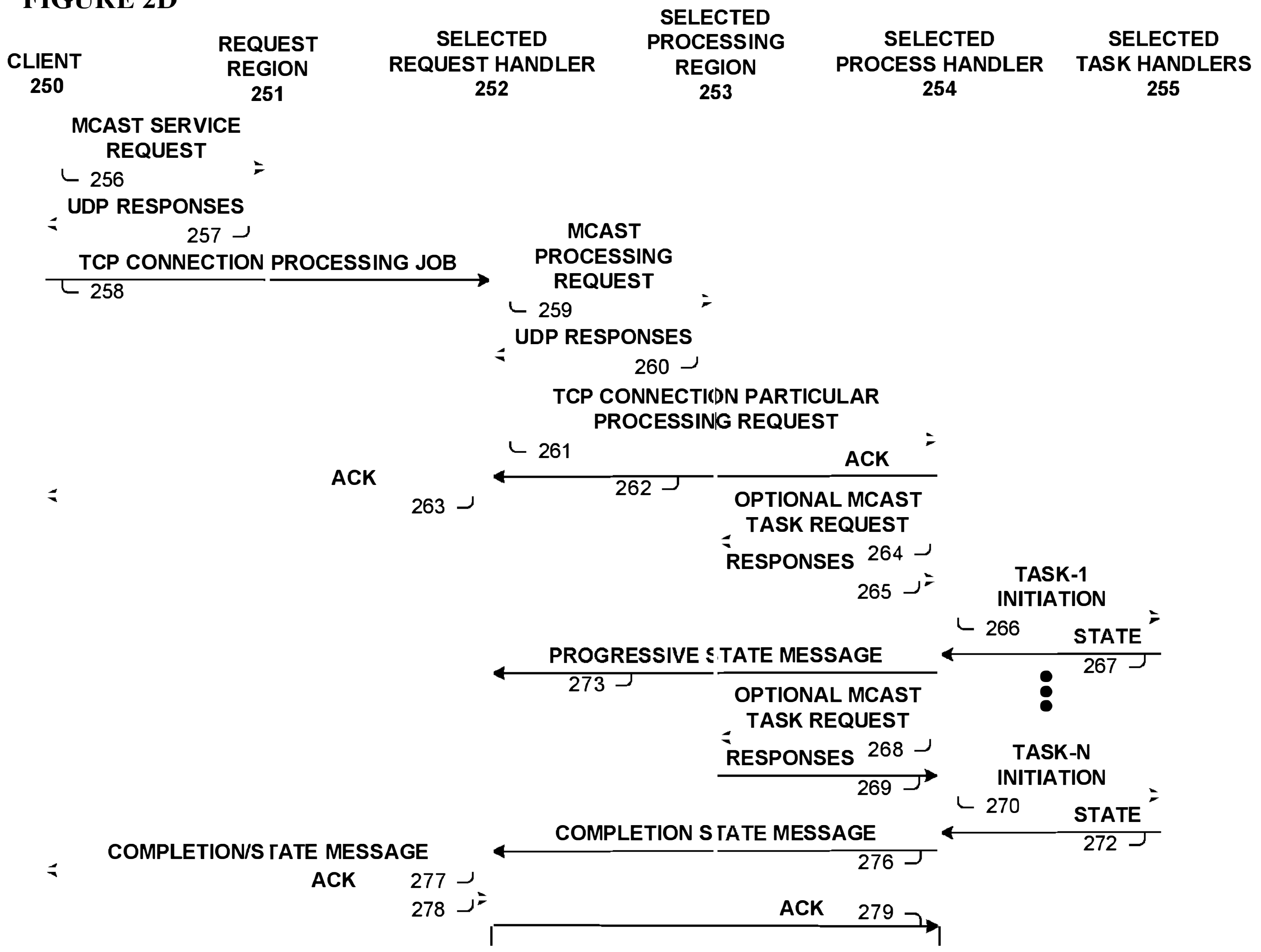

MULTICAST ADDRESSES
240
241
REQUEST REGION
242
PROCESSING REGION LEADER INTERCOMMUNICATION
243
PROCESSING REGION ACTIVE REGION INDICATIONS
244
PROCESSING REGION - 1
PROCESSING REGION - N

FIGURE 2C

FIGURE 2D


CLIENT 250
REQUEST REGION 251
SELECTED REQUEST HANDLER 252
SELECTED PROCESSING REGION 253
SELECTED PROCESS HANDLER 254
SELECTED TASK HANDLERS 255
MCAST SERVICE REQUEST
256
UDP RESPONSES
257
TCP CONNECTION PROCESSING JOB
258
MCAST PROCESSING REQUEST
259
UDP RESPONSES
260
TCP CONNECTION PARTICULAR PROCESSING REQUEST
261
ACK
262
ACK
263
OPTIONAL MCAST TASK REQUEST
RESPONSES 264
265
TASK-1 INITIATION
266
STATE
267
PROGRESSIVE STATE MESSAGE
273
OPTIONAL MCAST TASK REQUEST
RESPONSES 268
269
TASK-N INITIATION
270
STATE
272
COMPLETION STATE MESSAGE
276
COMPLETION/STATE MESSAGE
ACK 277
278
ACK 279

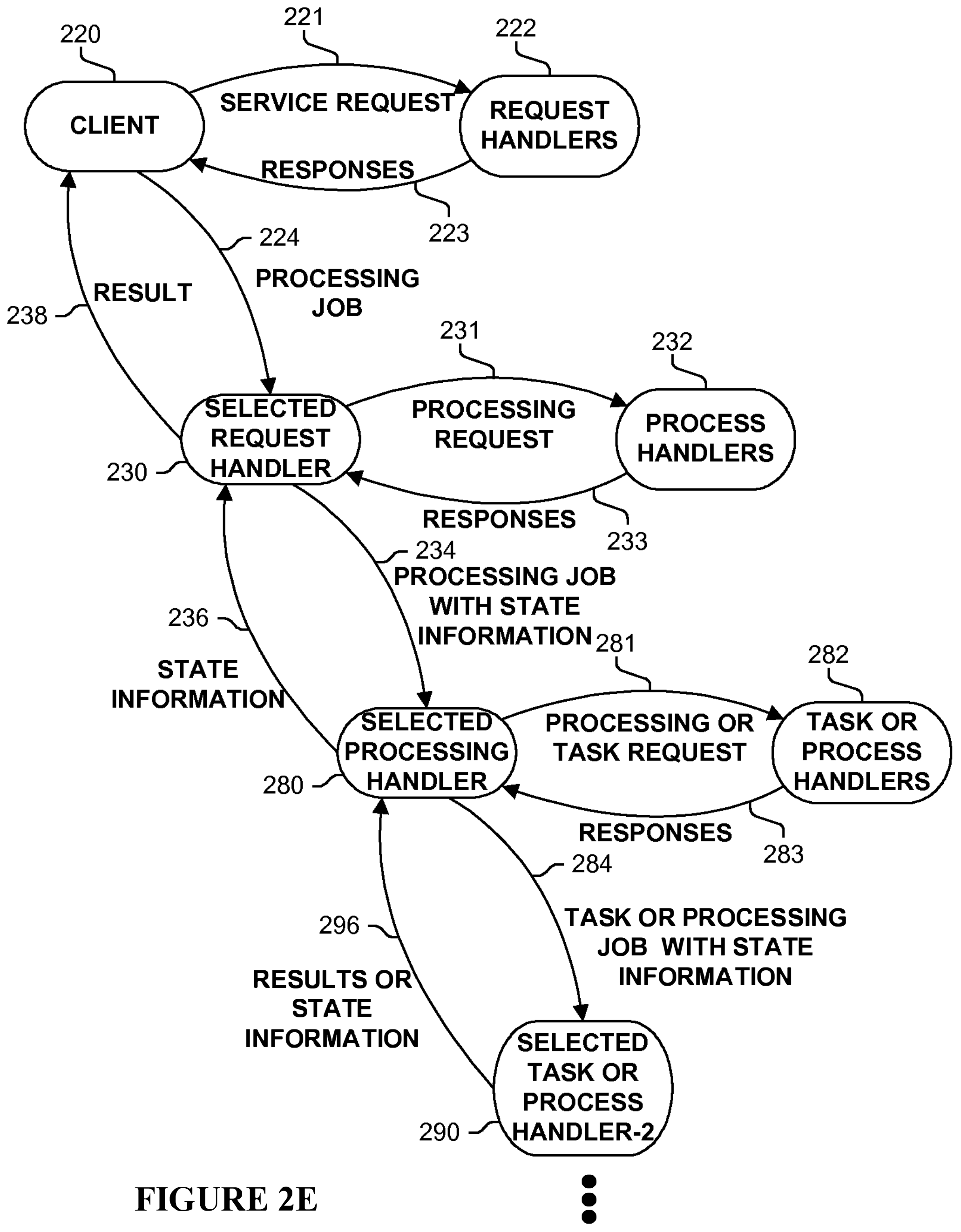
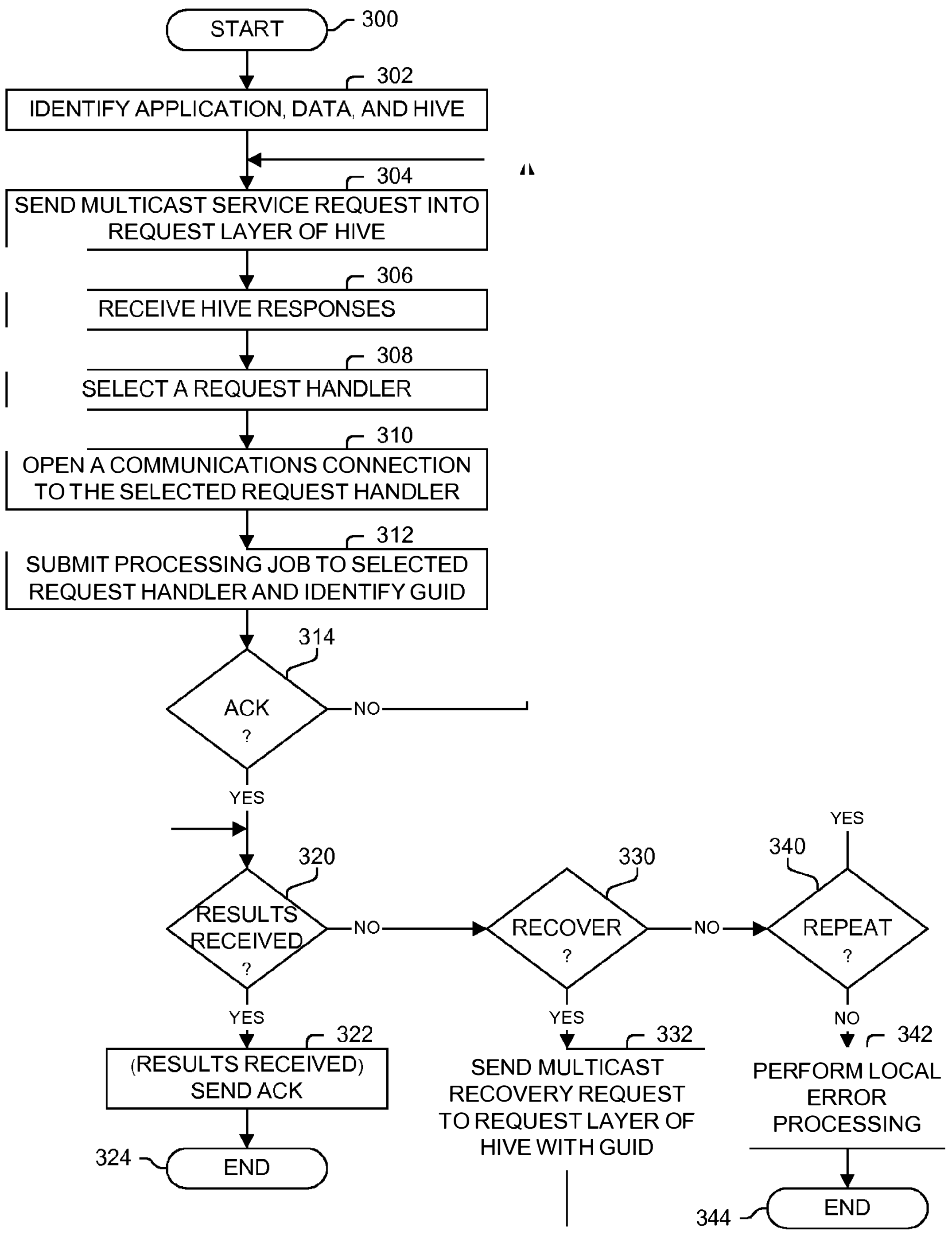
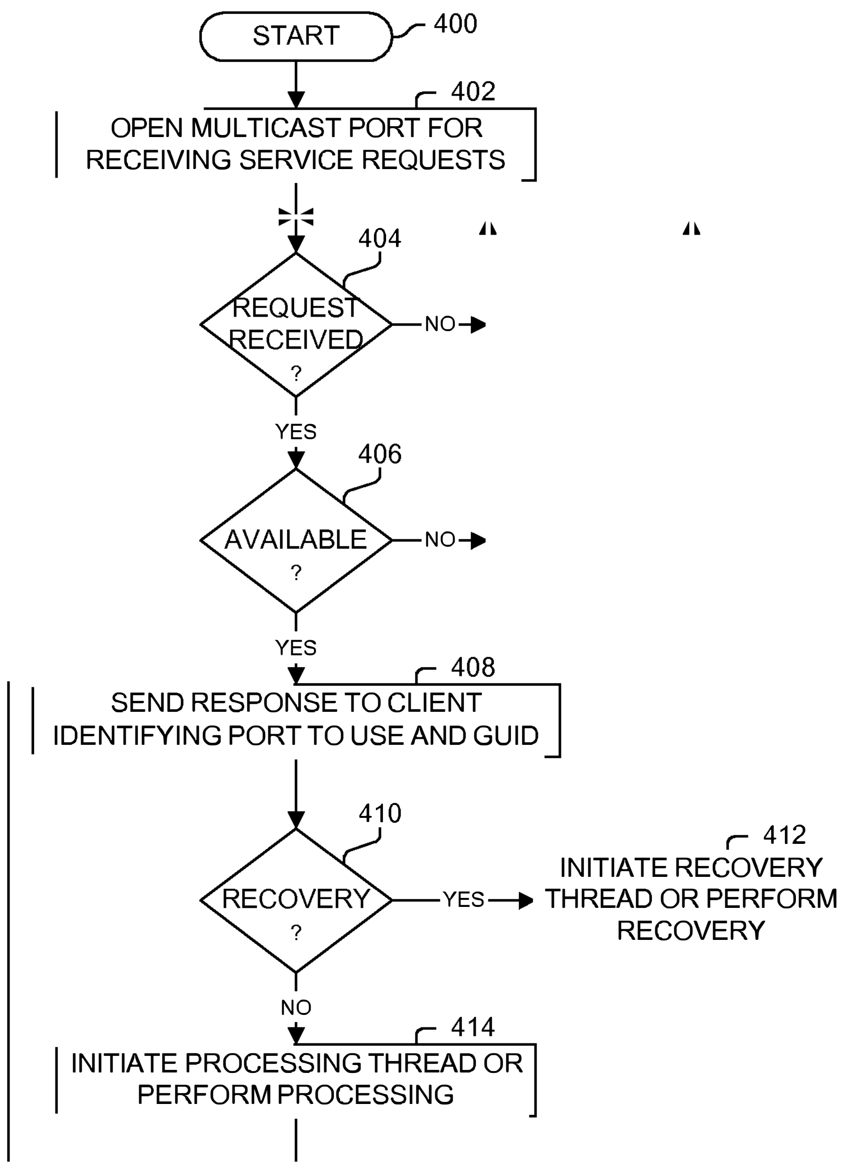

220
CLIENT
221
SERVICE REQUEST
222
REQUEST HANDLERS
RESPONSES
223
224
PROCESSING JOB
238
RESULT
231
PROCESSING REQUEST
232
PROCESS HANDLERS
SELECTED REQUEST HANDLER
230
RESPONSES
233
234
PROCESSING JOB WITH STATE INFORMATION
236
STATE INFORMATION
281
PROCESSING OR TASK REQUEST
282
TASK OR PROCESS HANDLERS
SELECTED PROCESSING HANDLER
280
RESPONSES
283
284
TASK OR PROCESSING JOB WITH STATE INFORMATION
296
RESULTS OR STATE INFORMATION
SELECTED TASK OR PROCESS HANDLER-2
290

FIGURE 2E


START
300
302
IDENTIFY APPLICATION, DATA, AND HIVE
304
SEND MULTICAST SERVICE REQUEST INTO REQUEST LAYER OF HIVE
306
RECEIVE HIVE RESPONSES
308
SELECT A REQUEST HANDLER
310
OPEN A COMMUNICATIONS CONNECTION TO THE SELECTED REQUEST HANDLER
312
SUBMIT PROCESSING JOB TO SELECTED REQUEST HANDLER AND IDENTIFY GUID
314
ACK ?
NO
YES
320
RESULTS RECEIVED ?
NO
YES
322
(RESULTS RECEIVED) SEND ACK
324
END
330
RECOVER ?
NO
YES
332
SEND MULTICAST RECOVERY REQUEST TO REQUEST LAYER OF HIVE WITH GUID
340
REPEAT ?
YES
NO
342
PERFORM LOCAL ERROR PROCESSING
344
END

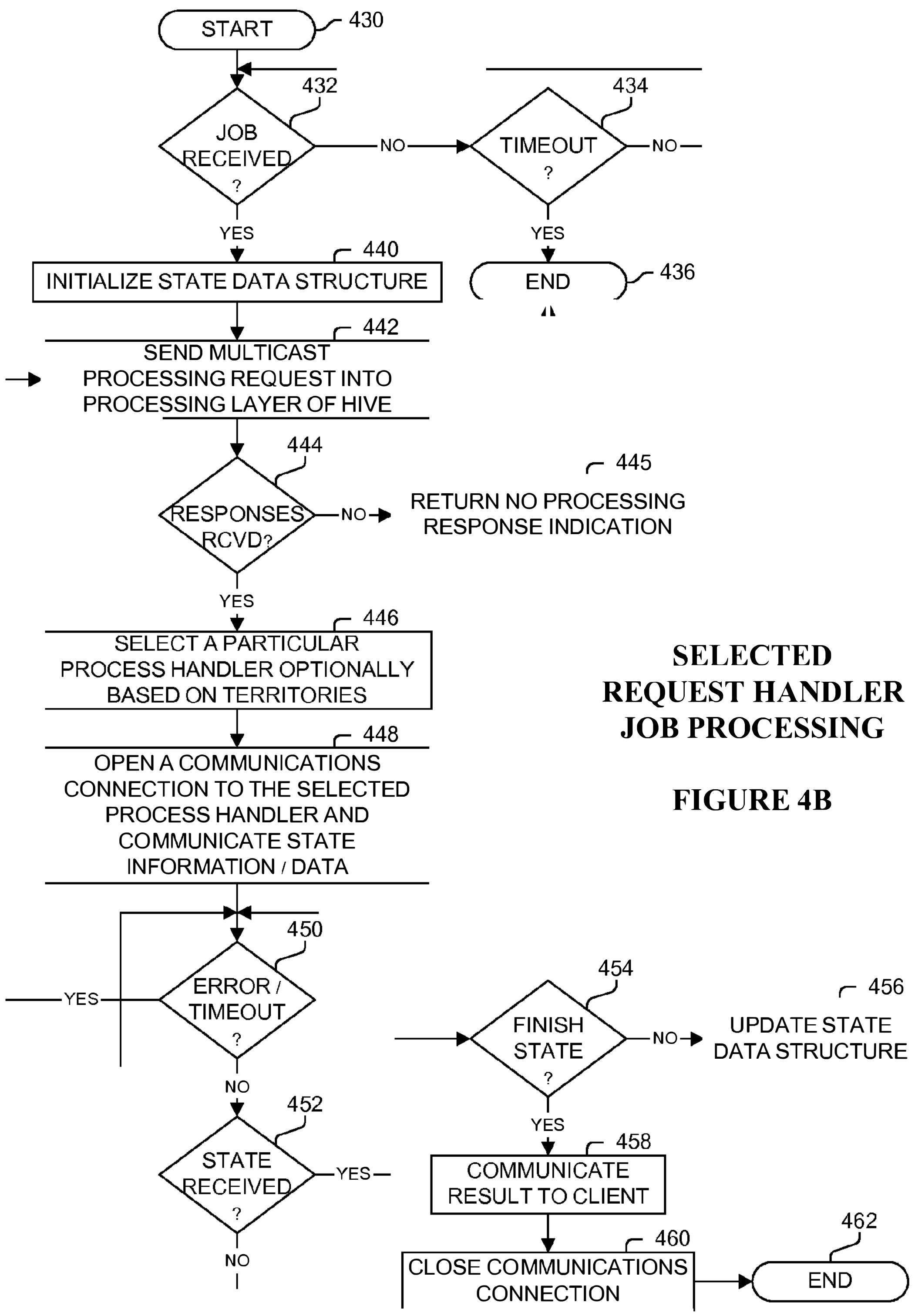
CLIENT PROCESSING

FIGURE 3


START
400
OPEN MULTICAST PORT FOR RECEIVING SERVICE REQUESTS
402
REQUEST RECEIVED ?
404
NO
YES
AVAILABLE ?
406
NO
YES
SEND RESPONSE TO CLIENT IDENTIFYING PORT TO USE AND GUID
408
RECOVERY ?
410
YES
INITIATE RECOVERY THREAD OR PERFORM RECOVERY
412
NO
INITIATE PROCESSING THREAD OR PERFORM PROCESSING
414

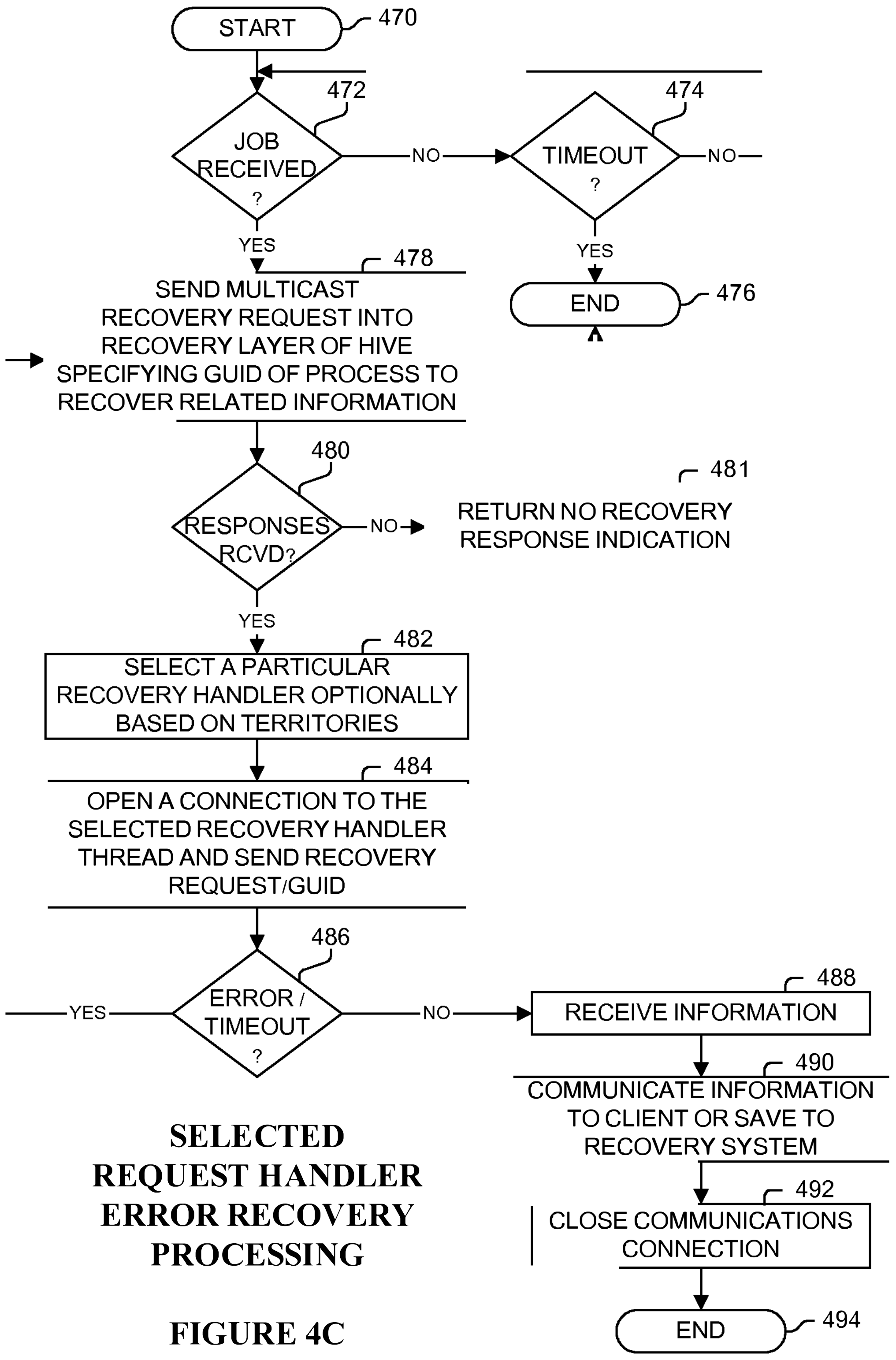
REQUEST HANDLER RESPONSE PROCESSING

FIGURE 4A


START
430
JOB RECEIVED ?
432
NO
TIMEOUT ?
434
NO
YES
YES
INITIALIZE STATE DATA STRUCTURE
440
END
436
SEND MULTICAST PROCESSING REQUEST INTO PROCESSING LAYER OF HIVE
442
RESPONSES RCVD?
444
NO
RETURN NO PROCESSING RESPONSE INDICATION
445
YES
SELECT A PARTICULAR PROCESS HANDLER OPTIONALLY BASED ON TERRITORIES
446
OPEN A COMMUNICATIONS CONNECTION TO THE SELECTED PROCESS HANDLER AND COMMUNICATE STATE INFORMATION / DATA
448
ERROR / TIMEOUT ?
450
YES
NO
STATE RECEIVED ?
452
YES
NO
FINISH STATE ?
454
NO
UPDATE STATE DATA STRUCTURE
456
YES
COMMUNICATE RESULT TO CLIENT
458
CLOSE COMMUNICATIONS CONNECTION
460
END
462

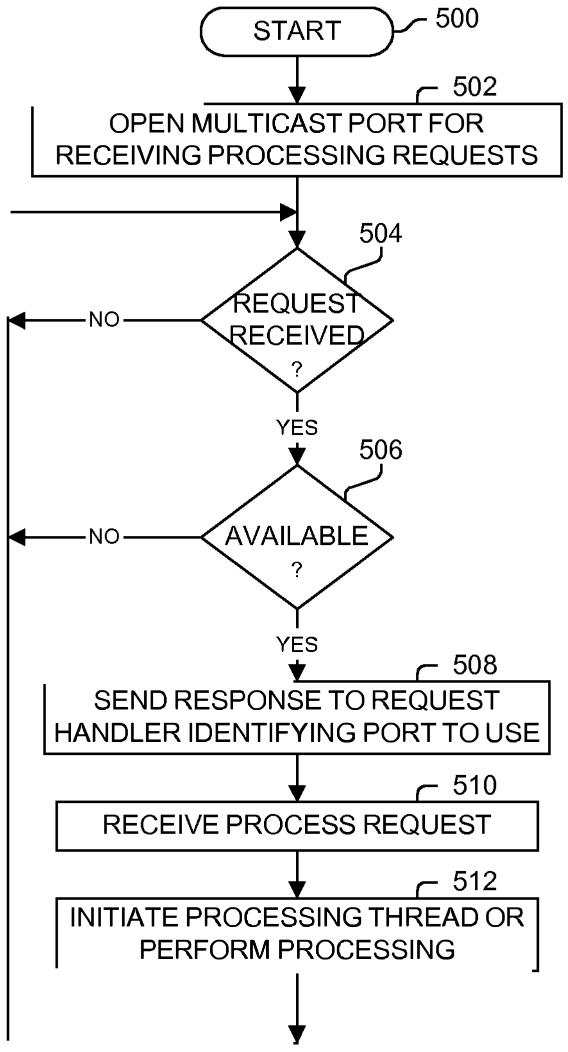
SELECTED REQUEST HANDLER JOB PROCESSING

FIGURE 4B


START
470
JOB RECEIVED ?
472
NO
TIMEOUT ?
474
NO
YES
YES
END
476
478
SEND MULTICAST RECOVERY REQUEST INTO RECOVERY LAYER OF HIVE SPECIFYING GUID OF PROCESS TO RECOVER RELATED INFORMATION
480
RESPONSES RCVD?
NO
481
RETURN NO RECOVERY RESPONSE INDICATION
YES
482
SELECT A PARTICULAR RECOVERY HANDLER OPTIONALLY BASED ON TERRITORIES
484
OPEN A CONNECTION TO THE SELECTED RECOVERY HANDLER THREAD AND SEND RECOVERY REQUEST/GUID
486
YES
ERROR / TIMEOUT ?
NO
488
RECEIVE INFORMATION
490
COMMUNICATE INFORMATION TO CLIENT OR SAVE TO RECOVERY SYSTEM
492
CLOSE COMMUNICATIONS CONNECTION
END
494

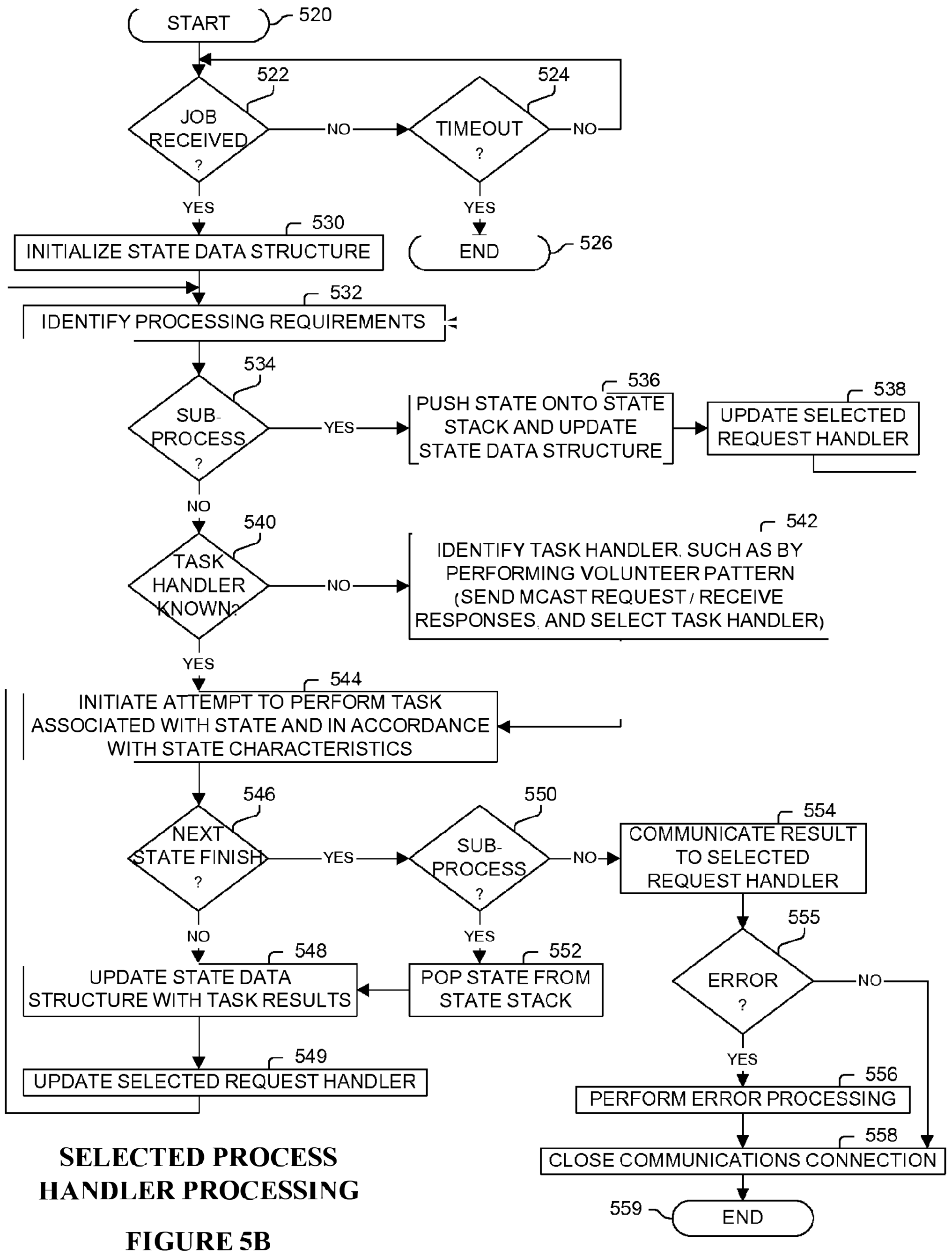
SELECTED REQUEST HANDLER ERROR RECOVERY PROCESSING

FIGURE 4C


START
500
OPEN MULTICAST PORT FOR RECEIVING PROCESSING REQUESTS
502
REQUEST RECEIVED ?
504
NO
YES
AVAILABLE ?
506
NO
YES
SEND RESPONSE TO REQUEST HANDLER IDENTIFYING PORT TO USE
508
RECEIVE PROCESS REQUEST
510
INITIATE PROCESSING THREAD OR PERFORM PROCESSING
512

PROCESS HANDLER RESPONSE PROCESSING

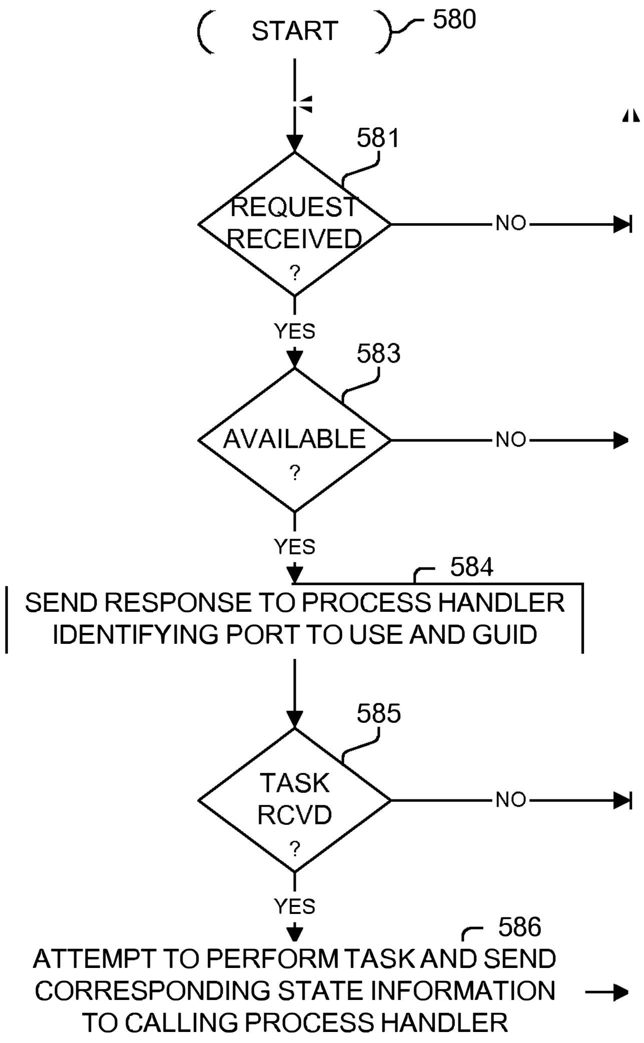
FIGURE 5A



START
520
522
JOB RECEIVED ?
NO
524
TIMEOUT ?
NO
YES
YES
530
INITIALIZE STATE DATA STRUCTURE
END
526
532
IDENTIFY PROCESSING REQUIREMENTS
534
SUB-PROCESS ?
YES
536
PUSH STATE ONTO STATE STACK AND UPDATE STATE DATA STRUCTURE
538
UPDATE SELECTED REQUEST HANDLER
NO
540
TASK HANDLER KNOWN?
NO
542
IDENTIFY TASK HANDLER, SUCH AS BY PERFORMING VOLUNTEER PATTERN (SEND MCAST REQUEST / RECEIVE RESPONSES, AND SELECT TASK HANDLER)
YES
544
INITIATE ATTEMPT TO PERFORM TASK ASSOCIATED WITH STATE AND IN ACCORDANCE WITH STATE CHARACTERISTICS
546
NEXT STATE FINISH ?
YES
550
SUB-PROCESS ?
NO
554
COMMUNICATE RESULT TO SELECTED REQUEST HANDLER
555
ERROR ?
NO
NO
YES
548
UPDATE STATE DATA STRUCTURE WITH TASK RESULTS
552
POP STATE FROM STATE STACK
YES
556
PERFORM ERROR PROCESSING
549
UPDATE SELECTED REQUEST HANDLER
558
CLOSE COMMUNICATIONS CONNECTION
559
END

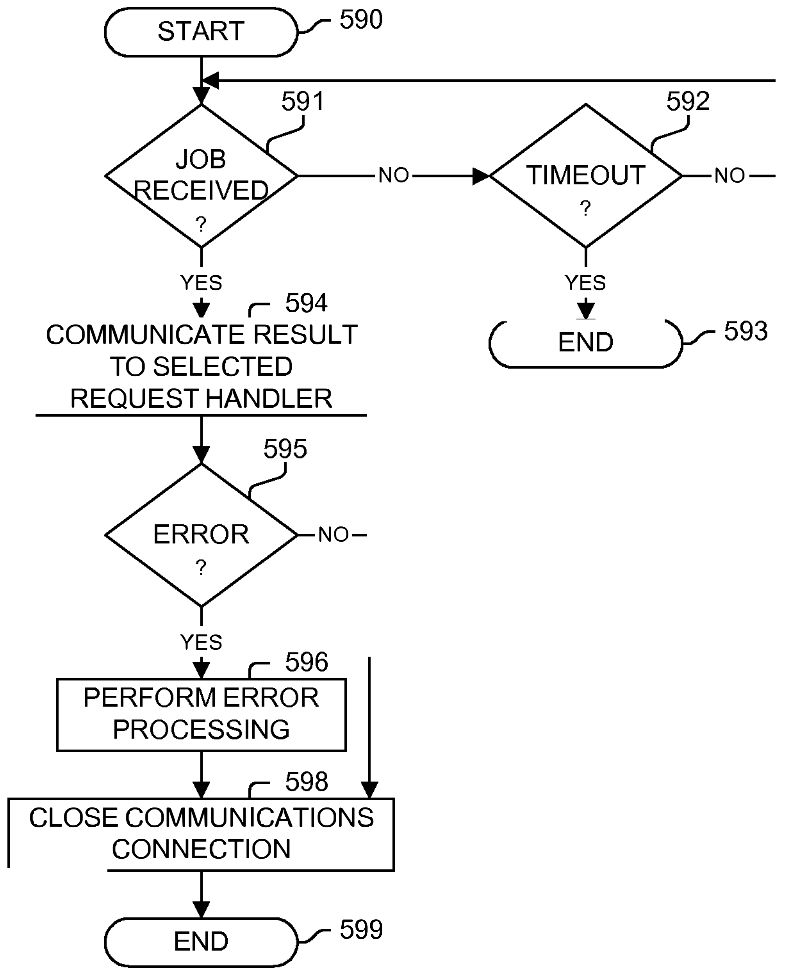
SELECTED PROCESS HANDLER PROCESSING

FIGURE 5B


START
580
581
REQUEST RECEIVED ?
NO
YES
583
AVAILABLE ?
NO
YES
584
SEND RESPONSE TO PROCESS HANDLER IDENTIFYING PORT TO USE AND GUID
585
TASK RCVD ?
NO
YES
586
ATTEMPT TO PERFORM TASK AND SEND CORRESPONDING STATE INFORMATION TO CALLING PROCESS HANDLER

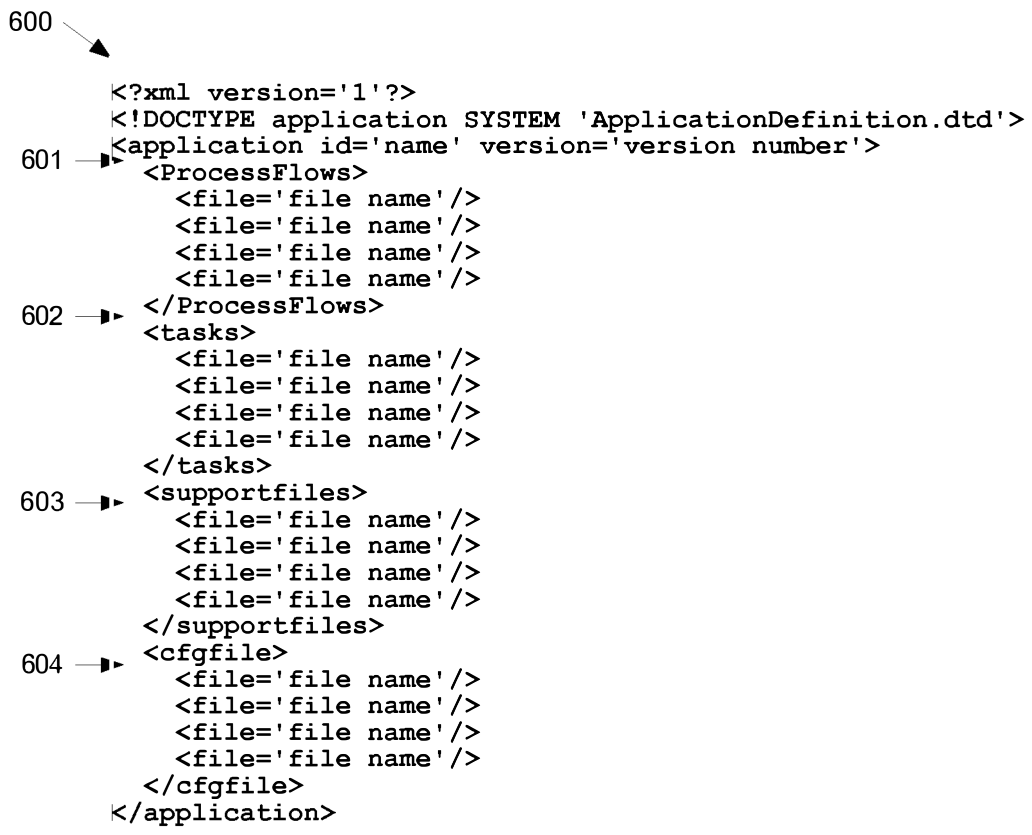
TASK HANDLER PROCESSING

FIGURE 5C



RECOVERY LAYER PROCESSING

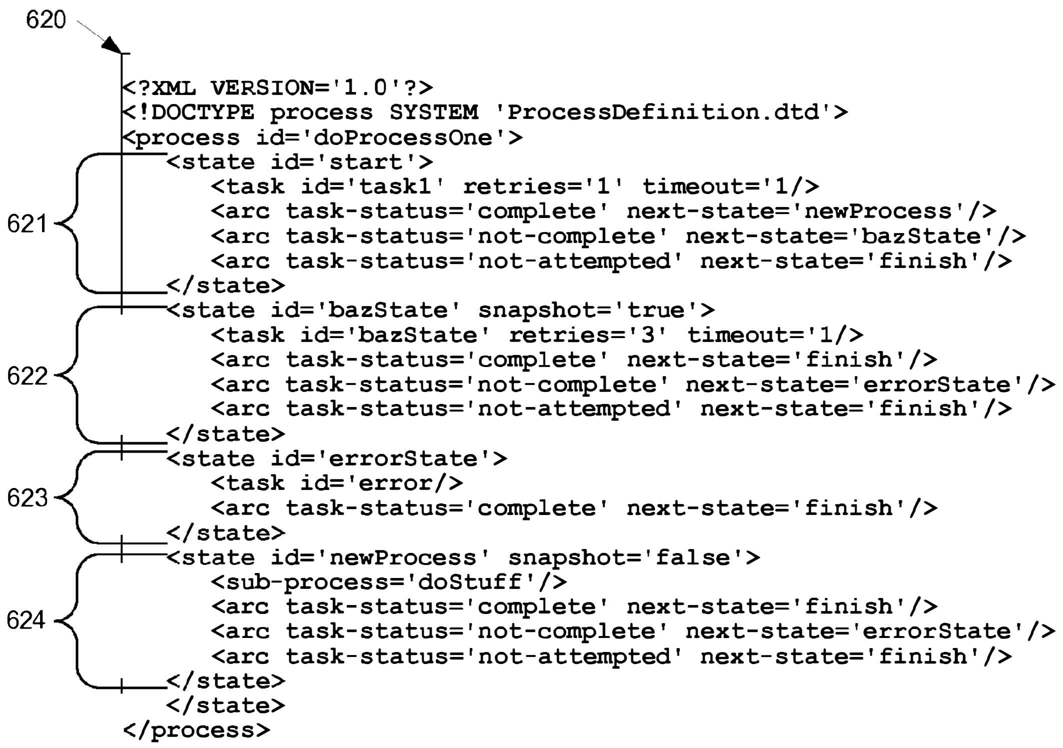
FIGURE 5D



600
601
602
603
604

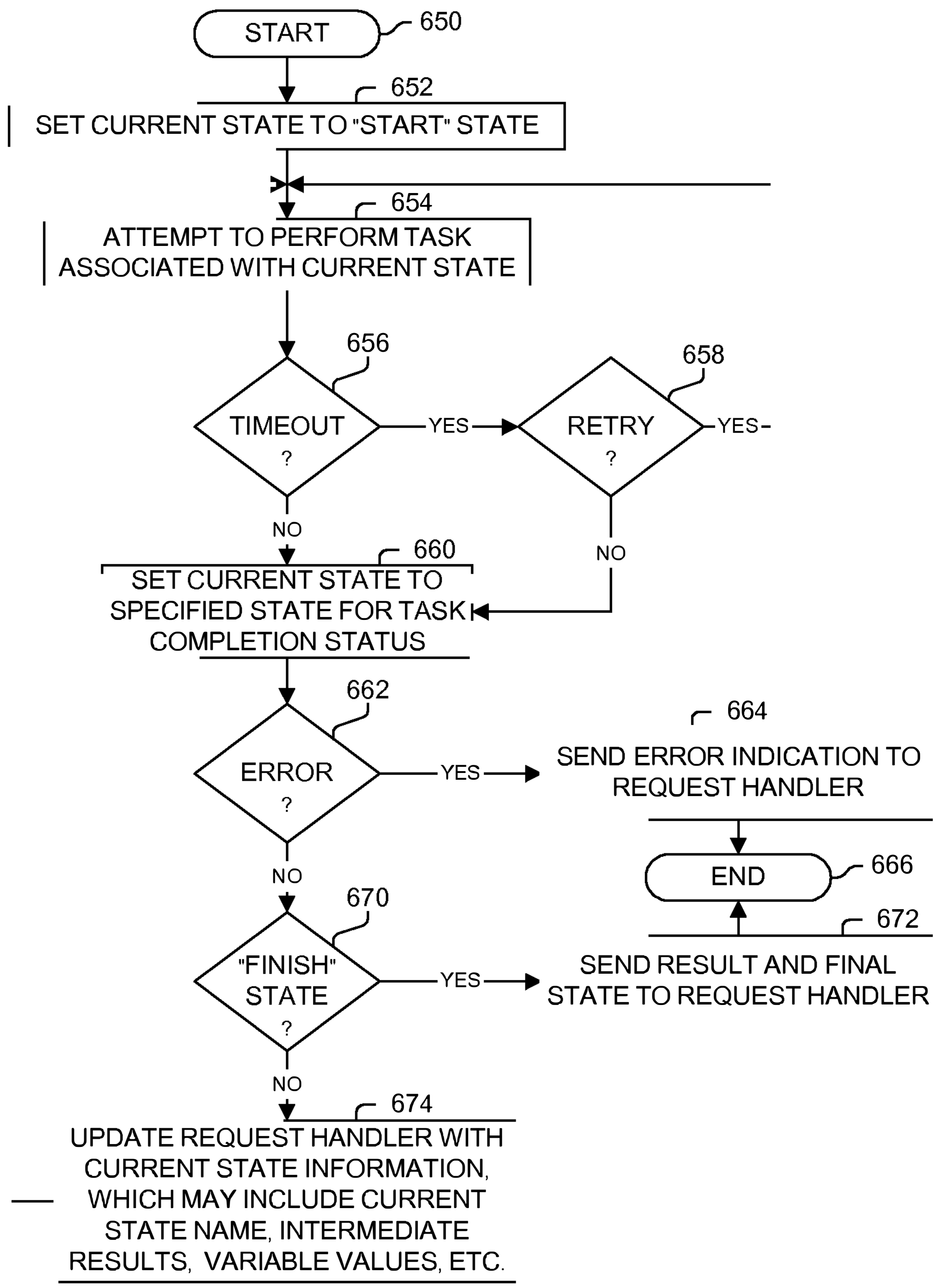
```
<?xml version='1'?>
<!DOCTYPE application SYSTEM 'ApplicationDefinition.dtd'>
<application id='name' version='version number'>
  <ProcessFlows>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </ProcessFlows>
  <tasks>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </tasks>
  <supportfiles>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </supportfiles>
  <cfgfile>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </cfgfile>
</application>
```

FIGURE 6A



```
<?XML VERSION='1.0'?>
<!DOCTYPE process SYSTEM 'ProcessDefinition.dtd'>
<process id='doProcessOne'>
   <state id='start'>
      <task id='task1' retries='1' timeout='1/>
      <arc task-status='complete' next-state='newProcess'/>
      <arc task-status='not-complete' next-state='bazState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   <state id='bazState' snapshot='true'>
      <task id='bazState' retries='3' timeout='1/>
      <arc task-status='complete' next-state='finish'/>
      <arc task-status='not-complete' next-state='errorState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   <state id='errorState'>
      <task id='error/>
      <arc task-status='complete' next-state='finish'/>
   </state>
   <state id='newProcess' snapshot='false'>
      <sub-process='doStuff'/>
      <arc task-status='complete' next-state='finish'/>
      <arc task-status='not-complete' next-state='errorState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   </state>
</process>
```

FIGURE 6B


START
650
652
SET CURRENT STATE TO "START" STATE
654
ATTEMPT TO PERFORM TASK ASSOCIATED WITH CURRENT STATE
656
TIMEOUT ?
YES
658
RETRY ?
YES
NO
NO
660
SET CURRENT STATE TO SPECIFIED STATE FOR TASK COMPLETION STATUS
662
ERROR ?
YES
664
SEND ERROR INDICATION TO REQUEST HANDLER
END
666
NO
670
"FINISH" STATE ?
YES
672
SEND RESULT AND FINAL STATE TO REQUEST HANDLER
NO
674
UPDATE REQUEST HANDLER WITH CURRENT STATE INFORMATION, WHICH MAY INCLUDE CURRENT STATE NAME, INTERMEDIATE RESULTS, VARIABLE VALUES, ETC.

EXEMPLARY PROCESS FLOW PROCESSING

FIGURE 6C

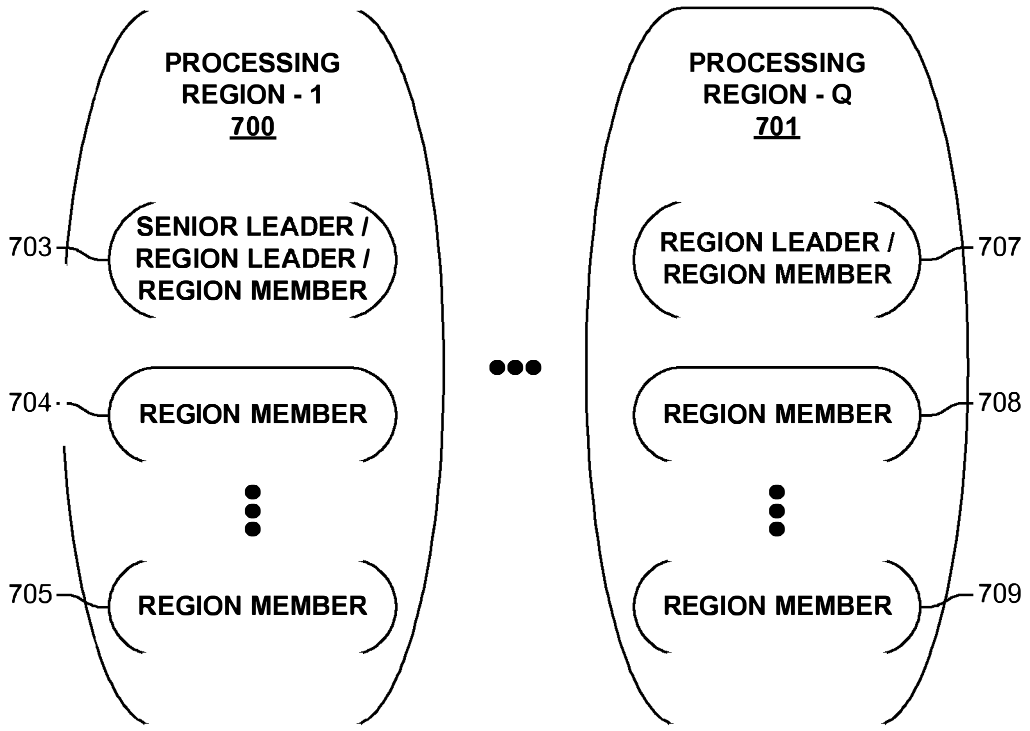
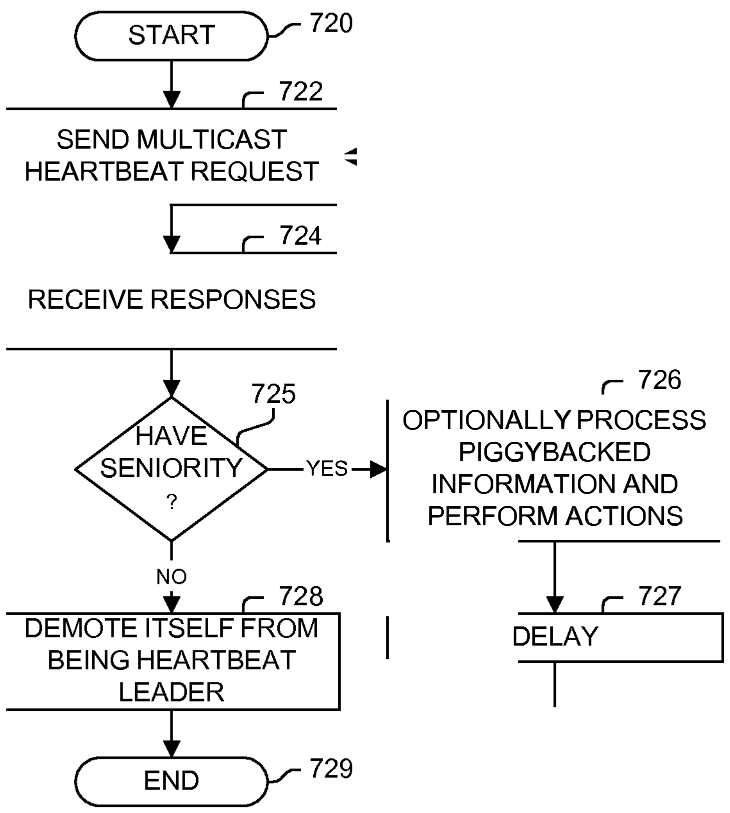
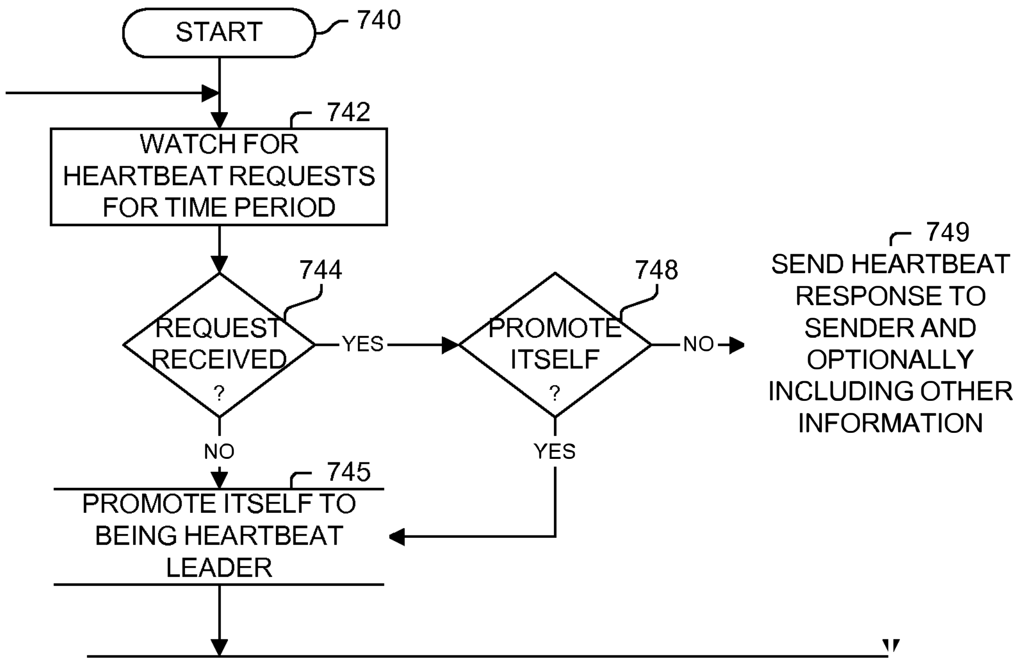

PROCESSING REGION - 1 700
SENIOR LEADER / REGION LEADER / REGION MEMBER
703
REGION MEMBER
704
REGION MEMBER
705
PROCESSING REGION - Q 701
REGION LEADER / REGION MEMBER
707
REGION MEMBER
708
REGION MEMBER
709

FIGURE 7A


START
720
722
SEND MULTICAST HEARTBEAT REQUEST
724
RECEIVE RESPONSES
725
HAVE SENIORITY ?
YES
726
OPTIONALLY PROCESS PIGGYBACKED INFORMATION AND PERFORM ACTIONS
NO
728
DEMOTE ITSELF FROM BEING HEARTBEAT LEADER
727
DELAY
END
729

HEARTBEAT LEADER

FIGURE 7B

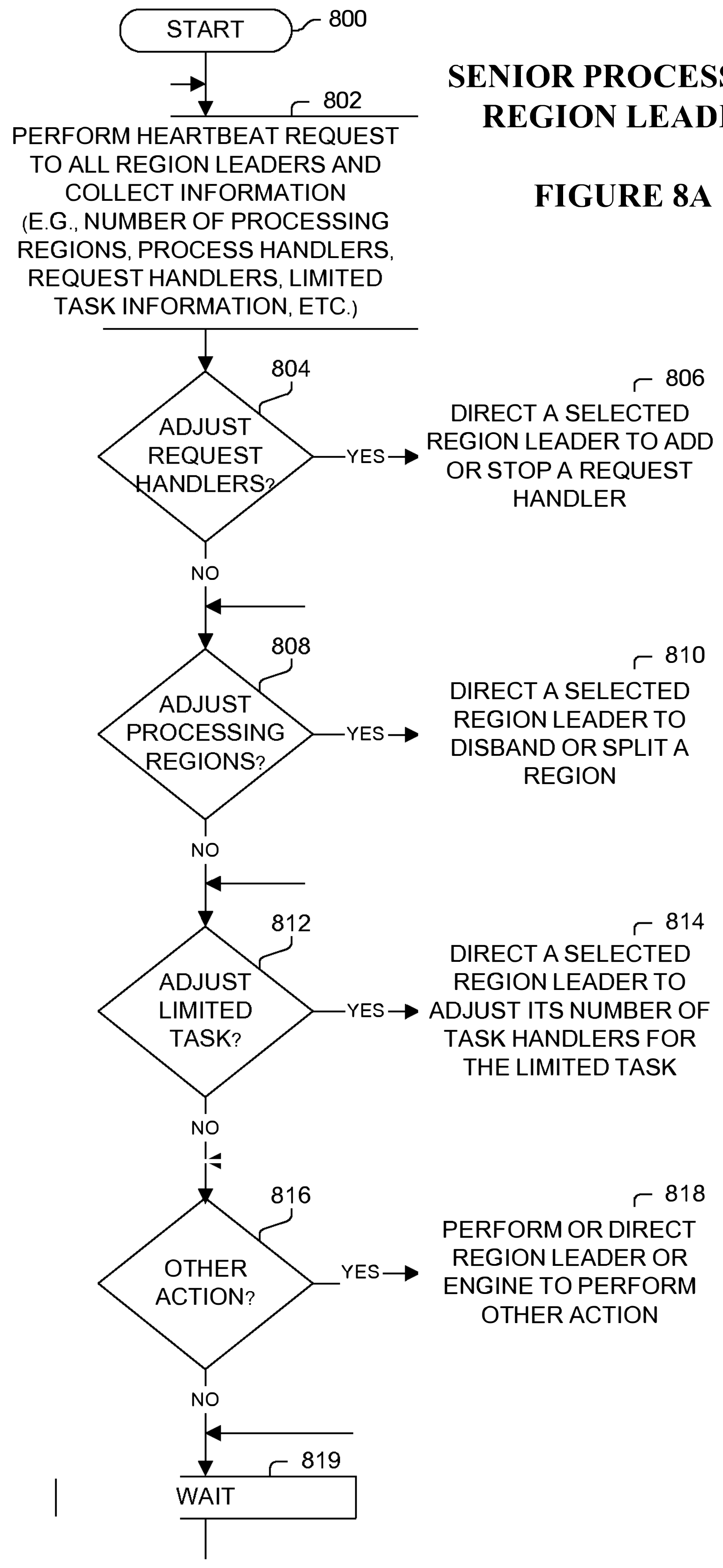

START
740
742
WATCH FOR HEARTBEAT REQUESTS FOR TIME PERIOD
744
REQUEST RECEIVED ?
YES
748
PROMOTE ITSELF ?
NO
749
SEND HEARTBEAT RESPONSE TO SENDER AND OPTIONALLY INCLUDING OTHER INFORMATION
NO
YES
745
PROMOTE ITSELF TO BEING HEARTBEAT LEADER

HEARTBEAT MEMBER

FIGURE 7C

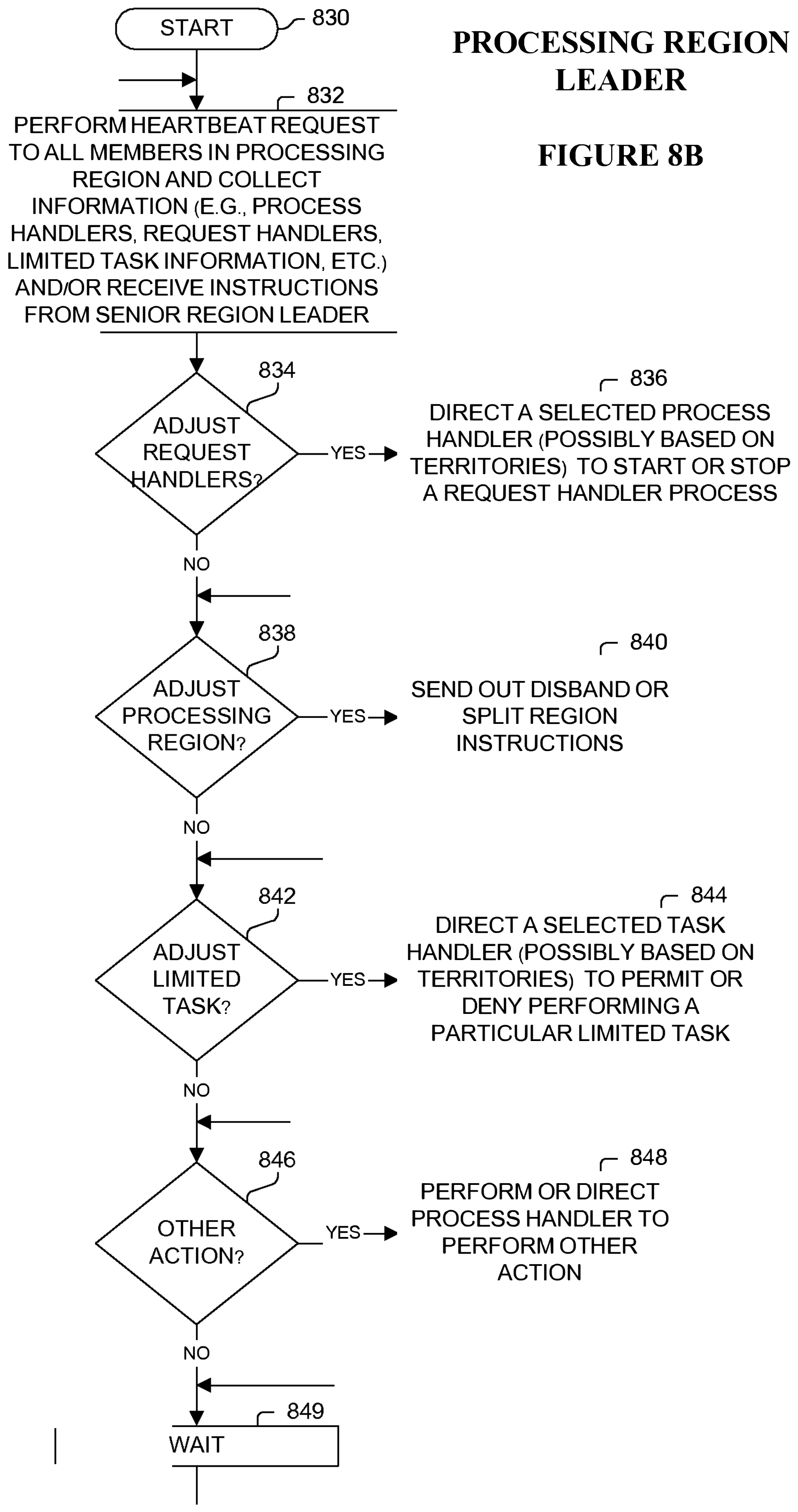
SENIOR PROCESSING REGION LEADER

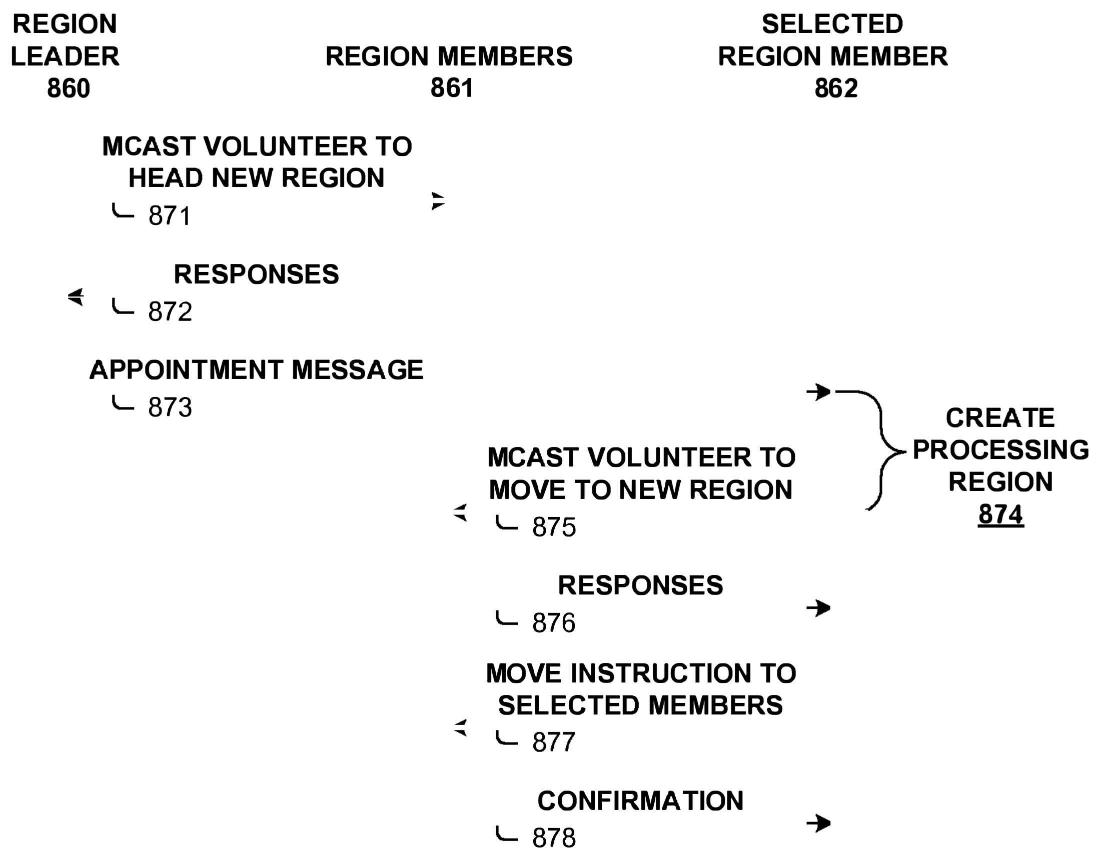
FIGURE 8A


START
800
802
PERFORM HEARTBEAT REQUEST TO ALL REGION LEADERS AND COLLECT INFORMATION (E.G., NUMBER OF PROCESSING REGIONS, PROCESS HANDLERS, REQUEST HANDLERS, LIMITED TASK INFORMATION, ETC.)
804
ADJUST REQUEST HANDLERS?
YES
806
DIRECT A SELECTED REGION LEADER TO ADD OR STOP A REQUEST HANDLER
NO
808
ADJUST PROCESSING REGIONS?
YES
810
DIRECT A SELECTED REGION LEADER TO DISBAND OR SPLIT A REGION
NO
812
ADJUST LIMITED TASK?
YES
814
DIRECT A SELECTED REGION LEADER TO ADJUST ITS NUMBER OF TASK HANDLERS FOR THE LIMITED TASK
NO
816
OTHER ACTION?
YES
818
PERFORM OR DIRECT REGION LEADER OR ENGINE TO PERFORM OTHER ACTION
NO
819
WAIT

PROCESSING REGION LEADER

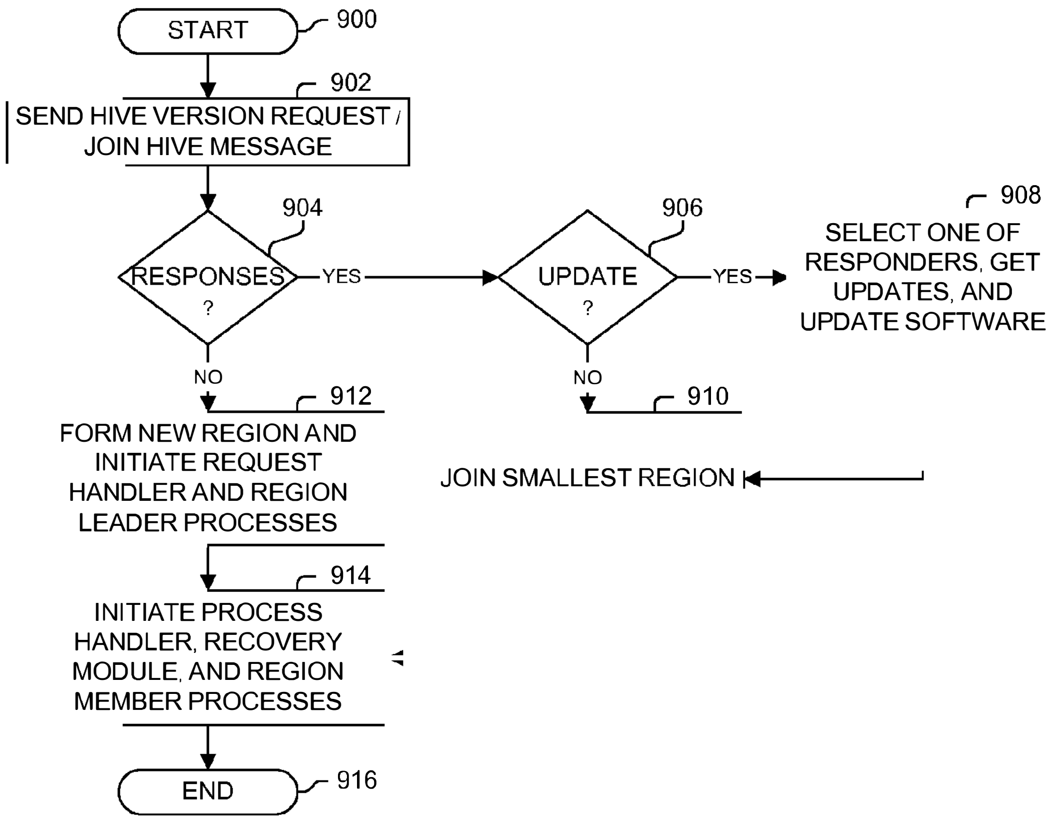
FIGURE 8B


START
830
832
PERFORM HEARTBEAT REQUEST TO ALL MEMBERS IN PROCESSING REGION AND COLLECT INFORMATION (E.G., PROCESS HANDLERS, REQUEST HANDLERS, LIMITED TASK INFORMATION, ETC.) AND/OR RECEIVE INSTRUCTIONS FROM SENIOR REGION LEADER
834
ADJUST REQUEST HANDLERS?
YES
836
DIRECT A SELECTED PROCESS HANDLER (POSSIBLY BASED ON TERRITORIES) TO START OR STOP A REQUEST HANDLER PROCESS
NO
838
ADJUST PROCESSING REGION?
YES
840
SEND OUT DISBAND OR SPLIT REGION INSTRUCTIONS
NO
842
ADJUST LIMITED TASK?
YES
844
DIRECT A SELECTED TASK HANDLER (POSSIBLY BASED ON TERRITORIES) TO PERMIT OR DENY PERFORMING A PARTICULAR LIMITED TASK
NO
846
OTHER ACTION?
YES
848
PERFORM OR DIRECT PROCESS HANDLER TO PERFORM OTHER ACTION
NO
849
WAIT


REGION LEADER 860
REGION MEMBERS 861
SELECTED REGION MEMBER 862
MCAST VOLUNTEER TO HEAD NEW REGION
871
RESPONSES
872
APPOINTMENT MESSAGE
873
CREATE PROCESSING REGION 874
MCAST VOLUNTEER TO MOVE TO NEW REGION
875
RESPONSES
876
MOVE INSTRUCTION TO SELECTED MEMBERS
877
CONFIRMATION
878

REGION SPLITTING

FIGURE 8C


START
900
902
SEND HIVE VERSION REQUEST / JOIN HIVE MESSAGE
904
RESPONSES ?
YES
906
UPDATE ?
YES
908
SELECT ONE OF RESPONDERS, GET UPDATES, AND UPDATE SOFTWARE
NO
912
FORM NEW REGION AND INITIATE REQUEST HANDLER AND REGION LEADER PROCESSES
NO
910
JOIN SMALLEST REGION
914
INITIATE PROCESS HANDLER, RECOVERY MODULE, AND REGION MEMBER PROCESSES
END
916

STARTUP PROCESS

FIGURE 9

SYSTEM AND METHOD FOR FAULT TOLERANT PROCESSING OF INFORMATION VIA NETWORKED COMPUTERS INCLUDING REQUEST HANDLERS, PROCESS HANDLERS, AND TASK HANDLERS

CROSS-REFERENCE AND PRIORITY CLAIM TO RELATED APPLICATIONS

This is a continuation of nonprovisional application Ser. No. 13/491,893, filed Jun. 8, 2012, now U.S. Pat. No. 8,341,209, which is a continuation of nonprovisional application Ser. No. 13/293,527, filed Nov. 10, 2011, now U.S. Pat. No. 8,200,746, which is a divisional of nonprovisional application Ser. No. 12/127,070, filed May 27, 2008, now U.S. Pat. No. 8,060,552, which is a divisional of nonprovisional application Ser. No. 10/236,784, filed Sep. 7, 2002, now U.S. Pat. No. 7,379,959, the entire disclosure of which being hereby incorporated by reference in its entirety.

FIELD OF THE INVENTION

This invention especially relates to processing of information including, but not limited to transactional processing using multiple networked computing systems; and more particularly, the invention relates to processing information using a hive of computing engines, typically including request handlers and process handlers.

INTRODUCTION

Many businesses are demanding faster, less expensive, and more reliable computing platforms. Brokerage houses, credit card processors, telecommunications firms, as well as banks are a few examples of organizations that require tremendous computing power to handle a countless number of small independent transactions. Currently, organizations that require these systems operate and maintain substantial servers. Further, the cost associated with these machines stems not only from the significant initial capital investment, but the continuing expense of a sizeable labor force dedicated to maintenance.

When it comes to mission-critical computing, businesses and other organizations face increasing pressure to do more with less. On one hand, they must manage larger transaction volumes, larger user populations, and larger data sets. They must do all of this in an environment that demands a renewed appreciation for the importance of reliability, fault tolerance, and disaster recovery. On the other hand, they must satisfy these growing requirements in a world of constrained resources. It is no longer an option to just throw large amounts of expensive hardware, and armies of expensive people, at problems. The challenge businesses face is that, when it comes to platforms for mission-critical computing, the world is fragmented. Different platforms are designed to satisfy different sets of requirements. As a result, businesses must choose between, and trade off, equally important factors.

Currently, when it comes to developing, deploying, and executing mission-critical applications, businesses and other organizations can choose between five alternative platforms. These are mainframes, high-availability computers, UNIX-based servers, distributed supercomputers, and PC's. Each of these approaches has strengths and weaknesses, advantages and disadvantages.

The first, and oldest, solution to the problem of mission-critical computing was the mainframe. Mainframes dominated the early days of computing because they delivered both availability and predictability. Mainframes deliver availability because they are located in extremely controlled physical environments and are supported by large cadres of dedicated, highly-trained people. This helps to ensure they do not fall victim to certain types of problems. However, because they are typically single-box machines, mainframes remain vulnerable to single-point failures. Mainframes deliver predictability because it is possible to monitor the execution and completion of processes and transactions and restart any that fail. However, the limitation of mainframes is that all monitoring code must be understood, written, and/or maintained by the application developer. The problem mainframes run into is that such systems fall short when it comes to three factors of high importance to businesses. First, mainframes tend not to offer high degrees of scalability. The only way to significantly increase the capability of such a system is to buy a new one. Second, because of their demanding nature, mainframes rely on armies of highly-trained support personnel and custom hardware. As a result, mainframes typically are neither affordable nor maintainable.

Developed to address the limitations and vulnerabilities of mainframes, high-availability computers are able to offer levels of availability and predictability that are equivalent to, and often superior to, mainframes. High-availability computers deliver availability because they use hardware or software-based approaches to ensure high levels of survivability. However, this availability is only relative because such systems are typically made up of a limited number of components. High-availability computers also deliver predictability because they offer transaction processing and monitoring capabilities. However, as with mainframes, that monitoring code must be understood, written, and/or maintained by the application developer. The problem with high-availability computers is that have many of the same shortcomings as mainframes. That means that they fall short when it comes to delivering scalability, affordability, and maintainability. First, they are largely designed to function as single-box systems and thus offer only limited levels of scalability. Second, because they are built using custom components, high-availability computers tend not to be either affordable or maintainable.

UNIX-based servers are scalable, available, and predictable but are expensive both to acquire and to maintain. Distributed supercomputers, while delivering significant degrees of scalability and affordability, fall short when it comes to availability. PC's are both affordable and maintainable, but do not meet the needs of businesses and other organizations when it comes to scalability, availability, and predictability. The 1990's saw the rise of the UNIX-based server as an alternative to mainframes and high-availability computers. These systems have grown in popularity because, in addition to delivering availability and predictability, they also deliver significant levels of scalability. UNIX-based servers deliver degrees of scalability because it is possible to add new machines to a cluster and receive increases in processing power. They also deliver availability because they are typically implemented as clusters and thus can survive the failure of any individual node. Finally, UNIX-based servers deliver some degree of predictability. However, developing this functionality can require significant amounts of custom development work.

One problem that UNIX-based servers run into, and the thing that has limited their adoption, is that this functionality comes at a steep price. Because they must be developed and maintained by people with highly specialized skills, they fall short when it comes to affordability and maintainability. For one thing, while it is theoretically possible to build a UNIX-based server using inexpensive machines, most are still implemented using small numbers of very expensive boxes. This makes upgrading a UNIX-based server an expensive and time-consuming process that must be performed by highly-skilled (and scarce) experts. Another limitation of UNIX-based servers is that developing applications for them typically requires a significant amount of effort. This requires application developers to be experts in both the UNIX environment and the domain at hand. Needless to say, such people can be hard to find and are typically quite expensive. Finally, setting up, expanding, and maintaining a UNIX-based server requires a significant amount of effort on the part of a person intimately familiar with the workings of the operating system. This reflects the fact that most were developed in the world of academia (where graduate students are plentiful). However, this can create significant issues for organizations that do not have such plentiful supplies of cheap, highly-skilled labor.

A recent development in the world of mission-critical computing is the distributed supercomputer (also known as a Network of Workstations or "NOW"). A distributed supercomputer is a computer that works by breaking large problems up into a set of smaller ones that can be spread across many small computers, solved independently, and then brought back together. Distributed supercomputers were created by academic and research institutions to harness the power of idle PC and other computing resources. This model was then adapted to the business world, with the goal being to make use of underused desktop computing resources. The most famous distributed supercomputing application was created by the Seti@Home project. Distributed supercomputers have grown in popularity because they offer both scalability and affordability. Distributed supercomputers deliver some degree of scalability because adding an additional resource to the pool usually yields a linear increase in processing power. However that scalability is limited by the fact that communication with each node takes place over the common organizational network and can become bogged down. Distributed supercomputers are also relatively more affordable than other alternatives because they take advantage of existing processing resources, be they servers or desktop PC's.

One problem distributed supercomputers run into is that they fall short when it comes to availability, predictability, and maintainability. Distributed supercomputers have problems delivering availability and predictability because they are typically designed to take advantage of non-dedicated resources. The problem is that it is impossible to deliver availability and predictability when someone else has primary control of the resource and your application is simply completing its work when it gets the chance. This makes distributed supercomputers appropriate for some forms of off-peak processing but not for time-sensitive or mission-critical computing. Finally, setting up, expanding, and maintaining a distributed supercomputer also requires a significant amount of effort because they tend to offer more of a set of concepts than a set of tools. As a result, they require significant amounts of custom coding. Again, this reflects the fact that most were developed in the world of academia where highly trained labor is both cheap and plentiful.

PC's are another option for creating mission-critical applications. PC's have two clear advantages relative to other solutions. First, PC's are highly affordable. The relentless progress of Moore's law means that increasingly powerful PC's can be acquired for lower and lower prices. The other advantage of PC's is that prices have fallen to such a degree that many people have begun to regard PC's as disposable. Given how fast the technology is progressing, in many cases it makes more sense to replace a PC than to repair it. Of course, the problem with PC's is that they do not satisfy the needs of businesses and other organizations when it comes to scalability, availability, and predictability. First, because PC's were designed to operate as stand-alone machines, they are not inherently scalable. Instead, the only way to allow them to scale is to link them together into clusters. That can be a very time-consuming process. Second, PC's, because they were designed for use by individuals, were not designed to deliver high levels of availability. As a result, the only way to make a single PC highly available is through the use of expensive, custom components. Finally, PC's were not designed to handle transaction processing and thus do not have any provisions for delivering predictability. The only way to deliver this functionality is to implement it using the operating system or an application server. The result is that few organizations even consider using PC's for mission-critical computing.

In a dynamic environment, it is important to be able to find available services. Service Location Protocol, RFC 2165, June 1997, provides one such mechanism. The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet no longer need so much static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration. The basic operation in Service Location is that a client attempts to discover the location of a Service. In smaller installations, each service will be configured to respond individually to each client. In larger installations, services will register their services with one or more Directory Agents, and clients will contact the Directory Agent to fulfill requests for Service Location information. Clients may discover the whereabouts of a Directory Agent by preconfiguration, DHCP, or by issuing queries to the Directory Agent Discovery multicast address.

The following describes the operations a User Agent would employ to find services on the site's network. The User Agent needs no configuration to begin network interaction. The User Agent can acquire information to construct predicates which describe the services that match the user's needs. The User Agent may build on the information received in earlier network requests to find the Service Agents advertising service information.

A User Agent will operate two ways. First, if the User Agent has already obtained the location of a Directory Agent, the User Agent will unicast a request to it in order to resolve a particular request. The Directory Agent will unicast a reply to the User Agent. The User Agent will retry a request to a Directory Agent until it gets a reply, so if the Directory Agent cannot service the request (say it has no information) it must return an response with zero values, possibly with an error code set.

Second, if the User Agent does not have knowledge of a Directory Agent or if there are no Directory Agents available on the site network, a second mode of discovery may be used. The User Agent multicasts a request to the service-specific multicast address, to which the service it wishes to locate will respond. All the Service Agents which are listening to this multicast address will respond, provided they can satisfy the User Agent's request. A similar mechanism is used for Directory Agent discovery. Service Agents which have no information for the User Agent MUST NOT respond.

While the multicast/convergence model may be important for discovering services (such as Directory Agents) it is the exception rather than the rule. Once a User Agent knows of the location of a Directory Agent, it will use a unicast request/response transaction. The Service Agent SHOULD listen for multicast requests on the service-specific multicast address, and MUST register with an available Directory Agent. This Directory Agent will resolve requests from User Agents which are unicasted using TCP or UDP. This means that a Directory Agent must first be discovered, using DHCP, the DA Discovery Multicast address, the multicast mechanism described above, or manual configuration. If the service is to become unavailable, it should be deregistered with the Directory Agent. The Directory Agent responds with an acknowledgment to either a registration or deregistration. Service Registrations include a lifetime, and will eventually expire. Service Registrations need to be refreshed by the Service Agent before their Lifetime runs out. If need be, Service Agents can advertise signed URLs to prove that they are authorized to provide the service.

New mechanisms for computing are desired, especially those which may provide a reliable computing framework and platform, including, but not limited to those which might produce improved levels of performance and reliability at a much lower cost than that of other solutions.

SUMMARY

A hive of computing engines, typically including request handlers and process handlers, is used to process information. One embodiment includes a request region including multiple request handlers and multiple processing regions, each typically including multiple process handlers. Each request handler is configured to respond to a client service request of a processing job, and if identified to handle the processing job: to query one or more of the processing regions to identify and assign a particular process handler to service the processing job, and to receive a processing result from the particular process handler. Each of the process handlers is configured to respond to such a query, and if identified as the particular process handler: to service the processing job, to process the processing job, to update said identified request handler with state information pertaining to partial processing of said processing job, and to communicate the processing result to the identified request handler. One embodiment includes multiple task handlers, wherein a process handler assigns a task identified with the processing job to one of task handlers, which performs the task and returns the result. In one embodiment, the selection of a task handler to perform a particular task is determined based on a volunteer pattern initiated by the process handler.

Another exemplary embodiment comprises a system for processing information, the system comprising a plurality of networked computers for processing a processing job in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job, the request handler configured to (1) receive a service request for the processing job, and (2) communicate data representative of the processing job to a process handler, the process handler to which the processing job data was communicated being configured to (1) receive the communicated processing job data, and (2) analyze the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers, the task handlers configured to (1) perform the processing tasks of the processing job in accordance with the determined sequence, and (2) generate updated state information in response to the

performed processing tasks, and wherein the request handler is further configured to (1) maintain state information for the processing job based on the updated state information, (2) determine whether a fault exists, and (3) in response to a determination that a fault exists, initiate a recovery procedure based on the maintained state information for the processing job.

Still another exemplary embodiment comprises a method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising: (1) receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (i) a plurality of processing tasks and (ii) state information relating to the processing job, (2) the request handler (i) receiving a service request for the processing job, and (ii) communicating data representative of the processing job to a process handler, (3) the process handler to which the processing job data was communicated (i) receiving the communicated processing job data, and (ii) analyzing the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers, (4) the task handlers (i) performing the processing tasks of the processing job in accordance with the determined sequence, and (ii) generating updated state information in response to the performed processing tasks, and (5) the request handler (i) maintaining state information for the processing job based on the updated state information, (ii) determining whether a fault exists, and (iii) in response to a determination that a fault exists, initiating a recovery procedure based on the maintained state information for the processing job.

Yet another exemplary embodiment comprises a method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising: (1) the request handler receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (i) a plurality of processing tasks and (ii) state information relating to the processing job, (2) a process handler coordinating an execution of the processing tasks by a plurality of the task handlers, (3) the task handlers (i) executing the processing tasks, and (ii) generating updated state information for the processing job in response to the executing step, (3) as the processing tasks are executed by the task handlers, redundantly storing updated state information across a plurality of different processes, (4) determining whether a failure has occurred, and (5) in response to a determination that a failure has occurred, (i) retrieving a copy of the redundantly stored state information, and (ii) resuming the processing job in accordance with the retrieved state information.

Yet another exemplary embodiment comprises a system for processing information, the system comprising a plurality of networked computers for processing a processing job in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job, the request handler configured to (1) receive a service request for the processing job, and (2) communicate data representative of the processing job to a process handler, the process handler to which the processing job data was communicated being configured to (1) receive the communicated processing job data, and (2) analyze the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers, the task handlers configured to (1) perform the processing tasks of the processing job in accordance with the determined sequence, and (2) generate updated state information in response to the performed processing tasks, and wherein the process handler to which the processing job data was communicated is further configured to (1) maintain state information for the processing job based on the updated state information, (2) determine whether a fault exists, and (3) in response to a determination that a fault exists, initiate a recovery procedure based on the maintained state information for the processing job.

Still another exemplary embodiment comprises a method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising: (1) receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (i) a plurality of processing tasks and (ii) state information relating to the processing job, (2) the request handler (i) receiving a service request for the processing job, and (ii) communicating data representative of the processing job to a process handler, (3) the process handler to which the processing job data was communicated (i) receiving the communicated processing job data, and (ii) analyzing the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers, (4) the task handlers (i) performing the processing tasks of the processing job in accordance with the determined sequence, and (ii) generating updated state information in response to the performed processing tasks, and (5) the process handler to which the processing job data was communicated (i) maintaining state information for the processing job based on the updated state information, (ii) determining whether a fault exists, and (iii) in response to a determination that a fault exists, initiating a recovery procedure based on the maintained state information for the processing job.

BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the invention with particularity. The invention, together with its advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

FIG. **1**A illustrates an architecture of hives used in one embodiment;

FIG. **1**B illustrates a computing platform used for a hive engine for implementing request handlers, process handlers, and/or other processes of a hive of one embodiment, or also used for simulating the operation of a hive in one embodiment;

FIG. **2**A illustrates a hierarchy of a hive, request regions, territories, and processing regions as used in one embodiment;

FIG. **2**B illustrates an interaction of a client, request handlers, and process handlers of one embodiment;

FIG. **2**C illustrates multicast addresses used in one embodiment;

FIG. **2**D illustrates the flow of messages between components of one embodiment;

FIG. **2**E illustrates an interaction of a client, request handlers, process handlers and possibly tasks of one embodiment;

FIG. **3** is a flow diagram of a client process used in one embodiment;

FIGS. **4**A-C are flow diagrams of request hander processes used in one embodiment;

FIG. **5**A-B are flow diagrams of process hander processes used in one embodiment;

FIG. **5**C is a flow diagram of a task handler process used in one embodiment;

FIG. **5**D is a flow diagram of a recovery layer process used in one embodiment;

FIG. **6**A illustrates a definition of an application used in one embodiment;

FIG. **6**B illustrates a definition of an process flow used in one embodiment;

FIG. **6**C illustrates a process used in one embodiment for executing a process flow;

FIG. **7**A illustrates a hierarchy of a senior region leaders, region leaders, and region members among multiple processing regions as used in one embodiment;

FIGS. **7**B-**7**C are flow diagrams of processes used in one embodiment to establish and maintain a hierarchical relationship among distributed processes;

FIG. **8**A is a flow diagram of a senior processing region leader process used in one embodiment;

FIG. **8**B is a flow diagram of a processing region leader process used in one embodiment;

FIG. **8**C illustrates the splitting of a region as performed in one embodiment; and

FIG. **9** illustrates a process used in one embodiment for initializing a hive engine.

DETAILED DESCRIPTION

A hive of computing engines, typically including request handlers and process handlers, is used to process information. Each of the claims individually recites an aspect of the invention in its entirety. Moreover, some embodiments described may include, but are not limited to, inter alia, systems, networks, integrated circuit chips, embedded processors, ASICs, methods, apparatus, and computer-readable medium containing instructions. The embodiments described hereinafter embody various aspects and configurations within the scope and spirit of the invention, with the figures illustrating exemplary and non-limiting configurations.

The term "system" is used generically herein to describe any number of components, elements, sub-systems, devices, packet switch elements, packet switches, routers, networks, computer and/or communication devices or mechanisms, or combinations of components thereof. The term "computer" is used generically herein to describe any number of computers, including, but not limited to personal computers, embedded processing elements and systems, control logic, ASICs, chips, workstations, mainframes, etc. The term "processing element" is used generically herein to describe any type of processing mechanism or device, such as a processor, ASIC, field programmable gate array, computer, etc. The term "device" is used generically herein to describe any type of mechanism, including a computer or system or component thereof. The terms "task" and "process" are used generically herein to describe any type of running program, including, but not limited to a computer process, task, thread, executing application, operating system, user process, device driver, native code, machine or other language, etc., and can be interactive and/or non-interactive, executing locally and/or remotely, executing in foreground and/or background, executing in the user and/or operating system address spaces, a routine of a library and/or standalone application, and is not limited to any particular memory partitioning technique. The steps, connections, and processing of signals and information illustrated in the figures, including, but not limited to any block and flow diagrams and message sequence charts, may be performed in the same or in a different serial or parallel ordering and/or by different components and/or processes, threads, etc., and/or over different connections and be combined with other functions in other embodiments in keeping within the scope and spirit of the invention. Furthermore, the term "identify" is used generically describe any manner or mechanism for directly or indirectly ascertaining something, which may included, but is not limited to receiving, retrieving from memory, determining, calculating, generating, etc.

Moreover, the terms "network" and "communications mechanism" are used generically herein to describe one or more networks, communications mediums or communications systems, including, but not limited to the Internet, private or public telephone, cellular, wireless, satellite, cable, local area, metropolitan area and/or wide area networks, a cable, electrical connection, bus, etc., and internal communications mechanisms such as message passing, interprocess communications, shared memory, etc. The term "message" is used generically herein to describe a piece of information which may or may not be, but is typically communicated via one or more communication mechanisms of any type, such as, but not limited to a packet.

As used herein, the term "packet" refers to packets of all types or any other units of information or data, including, but not limited to, fixed length cells and variable length packets, each of which may or may not be divisible into smaller packets or cells. The term "packet" as used herein also refers to both the packet itself or a packet indication, such as, but not limited to all or part of a packet or packet header, a data structure value, pointer or index, or any other part or identification of a packet. Moreover, these packets may contain one or more types of information, including, but not limited to, voice, data, video, and audio information. The term "item" is used herein to refer to a packet or any other unit or piece of information or data. The phrases "processing a packet" and "packet processing" typically refer to performing some steps or actions based on the packet, and which may or may not include modifying and/or forwarding the packet.

The term "storage mechanism" includes any type of memory, storage device or other mechanism for maintaining instructions or data in any format. "Computer-readable medium" is an extensible term including any memory, storage device, storage mechanism, and any other storage and signaling mechanisms including interfaces and devices such as network interface cards and buffers therein, as well as any communications devices and signals received and transmitted, and other current and evolving technologies that a computerized system can interpret, receive, and/or transmit. The term "memory" includes any random access memory (RAM), read only memory (ROM), flash memory, integrated circuits, and/or other memory components or elements. The term "storage device" includes any solid state storage media, disk drives, diskettes, networked services, tape drives, and other storage devices. Memories and storage devices may store computer-executable instructions to be executed by a processing element and/or control logic, and data which is manipulated by a processing element and/or control logic. The term "data structure" is an extensible term referring to any data element, variable, data structure, data base, and/or one or more or an organizational schemes that can be applied to data to facilitate interpreting the data or performing operations on it, such as, but not limited to memory locations or devices, sets, queues, trees, heaps, lists, linked lists, arrays, tables, pointers, etc. A data structure is typically maintained in a storage mechanism. The terms "pointer" and "link" are used generically herein to identify some mechanism for referencing or identifying another element, component, or other entity, and these may include, but are not limited to a reference to a memory or other storage mechanism or location therein, an index in a data structure, a value, etc.

The term "one embodiment" is used herein to reference a particular embodiment, wherein each reference to "one embodiment" may refer to a different embodiment, and the use of the term repeatedly herein in describing associated features, elements and/or limitations does not establish a cumulative set of associated features, elements and/or limitations that each and every embodiment must include, although an embodiment typically may include all these features, elements and/or limitations. In addition, the phrase "means for xxx" typically includes computer-readable medium containing computer-executable instructions for performing xxx.

In addition, the terms "first," "second," etc. are typically used herein to denote different units (e.g., a first element, a second element). The use of these terms herein does not necessarily connote an ordering such as one unit or event occurring or coming before the another, but rather provides a mechanism to distinguish between particular units. Additionally, the use of a singular tense of a noun is non-limiting, with its use typically including one or more of the particular item rather than just one (e.g., the use of the word "memory" typically refers to one or more memories without having to specify "memory or memories," or "one or more memories" or "at least one memory", etc.) Moreover, the phrases "based on x" and "in response to x" are used to indicate a minimum set of items x from which something is derived or caused, wherein "x" is extensible and does not necessarily describe a complete list of items on which the operation is performed, etc. Additionally, the phrase "coupled to" is used to indicate some level of direct or indirect connection between two elements or devices, with the coupling device or devices modify or not modifying the coupled signal or communicated information. The term "subset" is used to indicate a group of all or less than all of the elements of a set. Moreover, the term "or" is used herein to identify a selection of one or more, including all, of the conjunctive items.

Numerous means for processing information using a hive of computing/hive engines are disclosed. One implementation includes a request region including multiple request handlers and multiple processing regions, each typically including multiple process handlers. Each request handler is configured to respond to a client service request of a processing job, and if identified to handle the processing job: to query one or more of the processing regions to identify and assign a particular process handler to service the processing job, and to receive a processing result from the particular process handler. As typically used herein, a result corresponds to the outcome of a successfully or unsuccessfully completed job, task or other operation or an error condition, and typically includes one or more indications of a final value or outcome and/or state information (e.g., indications of to the processing performed or not performed, partial or final results, error descriptors, etc.) Each of the process handlers is configured to respond to such a query, and if identified as the particular process handler: to service the processing job, to process the processing job, to update said identified request handler with state information pertaining to partial processing of said processing job, and to communicate the processing result to the identified request handler.

In one embodiment, a volunteer pattern allows a software application (e.g., client process, request handler, process handler, task handler, tasks, or another hive engine process, etc.)

to automatically detect a group of software applications on the same network, and to select and communicate with the most appropriate application without any prior knowledge to the location and capabilities of the chosen software application. In one embodiment, messages are sent among processes typically using multicast UDP, unicast UDP, and standard TCP connections.

In one embodiment, the volunteer pattern includes the following steps. First, hive engines that wish to volunteer its capabilities begin by listening for volunteer requests on a known multicast address. Next, a client looking for a request handler to handle its request transmits its needs by issuing a volunteer or service request packet. The service request packet is a small text buffer which includes the type of service it is requesting and any potential parameters of that request. The service request packet also includes the return IP address of the client for hive engines to use to communicate their volunteer responses. The volunteer packet is communicated via multicast to the known multicast group corresponding to the request region. Request handlers of multiple hive engines on the client's network will detect this request. Third, hive engines that receive the service request packet examine its contents. If the hive engine is capable of servicing this request, it responds by sending a response (e.g., a UDP packet) to the client which made the request. The UDP packet typically contains the TCP address of the hive engine's communication port. Unicast UDP packets are used so that only the client that initiated the service request will receive the volunteer responses from the request handlers. Fourth, the client receives unicast UDP packets from the hive engines, selects one, and connects to the hive engine via TCP socket. The client and hive engine will typically use this socket for all subsequent communications during the processing of this application.

In one embodiment, regionalization is used to allow participating hive engines on the same network to detect each other and organize into logical groups of processing regions without any prior configuration to minimize bandwidth usage and CPU consumption in the entire system. Regionalization provides an automated mechanism that allows these processing regions grow and split as needed, which may provide for an unlimited growth of a hive. Thus, volunteer requests (e.g., processing requests, task requests, etc.) can be within a processing region without affecting all hive engines sending these requests or other communications using a multicast address assigned to a specific processing region. This places a bound on the number of responses to be generated (e.g., by the number of hive engines in a processing region.)

Typically, hive engines participate in an automated self-organization mechanisms, which allows participating hive engines on the same local or wide area network to detect each other and organize into logical groups without any prior configuration. However, an embodiment may use any mechanism for defining a regionalization, or even one embodiment does not use regionalization. For example, in one embodiment, a hive engine is pre-configured with parameters to define which region or regions in which to participate; while in one embodiment, users or a centralized control system is used to specify to one or more hive engines which region or regions in which to participate.

A hive typically has multiple processing regions and a single request region; although, one embodiment includes multiple request regions and one or more processing regions. One way to view a processing region is that it is a set of processes on one or more hive engines for executing processing jobs. In one embodiment, a processing region has a leader that keeps track of the number of hive engines in the region. If

the number of hive engines in the region reaches the user defined maximum, the region leader instructs the hive engines in the region to divide into two separate smaller regions. If the number of hive engines in the regions reaches the user defined minimum, the region leader instructs the hive engines in the region to join other regions in the hive.

In one embodiment, the processing regions are self-healing in that if the region leader shuts down for any reason all the region members detect the lack of a region leader. A region member promotes itself to region leader. If a processing region has multiple region leaders, the youngest region leaders demotes themselves back to region members, leaving one region leader.

A request region typically hides that the hive consists of multiple regions and directs the processing load across all the regions. From one perspective, spreading the request region across multiple hive engines provides an increased level of fault tolerance, as these services detect the loss of a connection and rebuild or shutdown as necessary. The hive recovers most failure cases, however, when a request is in an indeterminate state, the request is typically terminated to prevent multiple executions.

In one embodiment, a single senior region leader forms the request region. The senior region leader discovers the region leaders via the volunteer pattern. The senior region leader discovers the size of the request region by asking the region leaders for the number of hive engines in their region that are also members of the request region. If the request region has too many or too few members, the senior region leader directs the region leaders to re-allocate the hive engines to or from the request region. The request region is typically self-healing in that if the senior region leader shuts down for any reason all the region leaders detect the lack of a senior region leader. A region leader promotes itself to senior region leader. If the new senior region leader is not the most senior region leader, the senior region leader demotes itself and the most senior region leader promotes itself to senior region leader. If more than one senior region leader exists, the senior region leaders that are less senior or junior to another senior region leader demotes itself.

In one embodiment, a client processing job is specified in terms of a process flow, typically specifying a set of tasks as well state variables typically before and after each task for storing state information. The hive process flow contains the information on the sequence of sub-routines to be called, timeout and retry information if the sub-routines fail, and which sub-routine to call next based on the sub-routine's result. Once specified, it is up to the hive software to execute the sub-routines in the process flow. A process flow may described in any manner or format. For example, in one embodiment, a process flow is described in a XML process definition file. The process flow definition file defines the process flow name, the task to be performed, the task's recovery procedure including the timeout limit and retry limit, and the transition from one state to the next state based on the previous task's result.

In order to maintain high-availability and fault tolerance, a client processing job is typically performed using a self-organized, non-administered, network of services across several hive engines that work together to guarantee execution of a request even in the event that any of the individual services or hive engines fail. For example, in one embodiment, a processing job is received by a request handler from a client using the volunteer pattern. The request engine selects a process handler based on pattern. The process handler proceeds to perform the processing job, and at intermediate steps within the process flow, the process handler communicates state information to the request engine, such that the state and progress of the processing job at discrete steps is known by multiple processes, typically on different physical hive engines, and possibly in different territories (which may be defined to be in physically different locations, or using different communications and/or electrical systems, etc.) Thus, should a failure occur, the processing job typically can be resumed by another process handler newly selected by the request handler, or possibly completed by the original process handler with it storing results and/or communicating the results to the client via a different path (e.g., using a different request handler, etc.)

In one embodiment, processing a request typically includes the request setup, request processing, and request teardown. In the request setup, the client submits a request for a volunteer to the request region. A request handler receives the request, opens a TCP connection, and sends a response to the client. The client sends the request over the TCP connection to the request handler. The request handler receives the request and submits a request for a volunteer. A process handler receives the request, opens a TCP connection, and sends a response to the request handler. The request handler receives the response and sends the request over the TCP connection to the process handler. The process handler receives the request and sends an acknowledgement message. The request handler receives the acknowledgement message then sends an acknowledgement message to the client. The client receives the acknowledgement message then sends a process command to the request handler. The request handler receives the process command sends the process command to the process handler. The process handler receives the process command and begins processing the request. If the client loses connection with the request handler during this procedure, the client should perform a retry.

In one embodiment, in the request process procedure, the process handler submits a volunteer request to a processing region. A task handler receives the volunteer request, opens a TCP connection, and sends a response. The process handler receives the volunteer response and sends the first task in the process flow to the task handler over the TCP connection. The task handler processes the task and sends the results to the process handler. If the task does not complete within the specified amount of time and retries are set to zero, the request handler returns an error code as the final result to the request handler. If the task does not complete within the specified amount of time and retries are greater than zero, the request handler resubmits the task to another task handler. If snapshot is enabled on this task or if retries is set to zero, the process handler sends the result to the request handler. This repeats until the next state is finish. When the next state is finish, the process handler sends the final result to the request handler. If the client loses connection with the request handler during this procedure, the client should perform a recover.

In one embodiment, in the request teardown procedure, the request handler sends the final result to the client. The client receives the result and sends an acknowledgement to the request handler. The request handler receives the acknowledgement and sends an acknowledgement to the process handler. If the client loses connection with the request handler during this procedure, the client should perform a recover.

In one embodiment, the task service runs on each worker machine. Task services have an IP address and assigned TCP port on their worker machine. All task services in the Hive share common UDP multicast groups based on their worker machine's current region. On completion of the volunteer pattern for a simple task, the connected TCP socket will be passed off to the task handler. When responding to a volunteer pattern for a daemon task, this service will UDP the daemon task's IP and port to the requester. The service has both task handlers and daemon tasks. Upon receiving a task to execute from a process handler, the service will spin off a task handler or delegate the task to a daemon task, as appropriate. Upon completion of the task, the task handler or daemon task will return the results to the process handler.

One embodiment uses an intra-process recovery which enables the hive to recover from a connection loss between the client and the request handler while the request handler is overseeing the processing of a request. When the client loses the connection with a first request handler, once the request processing has completed the request setup phase, the first request handler continues processing the request and the client submits a request for a new request handler (second request handler). The client issues the recover command and second request handler listens queries the recover service for a user-defined amount of time. If second request handler does not receive the result within the specified amount of time, second request handler returns an error. When first request handler receives the final result, first request handler writes the final result to the recover service.

One embodiment operates slightly differently as multiple process handlers are used for each step in a process flow. For example, bothprocess handlers typically maintain the current state of the request such that if either of the process handlers is lost, the other picks up in its place. If the request handler is lost, the client and/or process handlers can establish a new request handler. The request handler manages the interface between software requesting processing from the hive and the hive. A primary process handler is a service that walks a request through the steps and recovery defined in a process flow. A secondary process handler is a service that monitors the primary process handler. If something happens to the primary process handler, the secondary process handler continues going through the steps and recovery defined in a process flow. A task handler is a service that performs the sub-routine defined in the process flow.

For example, in one embodiment, first, a request handler finds two process handlers. The request handler designates one as the primary process handler and the other as the secondary process handler. Next, the request handler sends the primary process handler the secondary process handler's IP address and sends the secondary process handler the primary process handler's IP address. The primary process handler and secondary process handler open a TCP port for communication then send acknowledgement messages to the request handler. The primary process handler finds a task handler. The task handler opens a TCP port and sends the request to the primary process handler. The primary process handler prepares the initial process flow state and sends that state to the secondary process handler. The secondary process handler and the request handler monitor the task states over the TCP connection. The task handler processes the request, sends the result to the primary process handler.

One embodiment provides an assimilation mechanism which recognizes new hive engines trying to join a hive. These steps occur without stopping execution of the entire hive, and he hive updates its hive engines in a measured rate to ensure that portions of the hive are continually processing requests ensuring constant availability of the hive applications.

In one embodiment, when a new hive engine joins the hive, the new hive engine finds the operating system image and the base hive software via DHCP. The new hive engine self installs the OS image and hive software using automated scripts defined by client. If a hive engine has an old version of the OS, the region leader makes the hive engine unavailable for processing. The hive engine is erased and rebooted. The hive engine then joins the hive as a new hive engine and re-installs the OS and hive software accordingly.

In addition, in one embodiment, when a hive engine joins the hive, the hive engine sends a request to the region leader. The hive engine receives a response from the region leader and selects a region to join. The region leader queries the hive engine for information about services, software, and versions. If the region leader is running a newer version of the hive system, the region leader makes the hive engine unavailable for processing. The region leader updates the hive engine by transmitting the current version of the hive system. The hive engine installs the update and commences processing. If the hive engine is running a newer version of hive system than the region leader, the region leader makes itself unavailable for process, receives the newer version of the hive system from the hive engine, installs the software, and continues processing. Once the region leader is updated, the region leader begins updating its region's members and the other region leaders. For example, in one embodiment, a hive engine then receives a response from the region leaders and selects a region to join. The region leader queries the hive engine for information about services, software, and versions. If the region leader is running the most current version of the hive applications, the region leader automatically updates the hive engine's hive applications. If the hive engine is running the most current version of the hive applications, the region leader automatically updates its hive applications. Once the region leader is updated, the region leader begins updating its region's members and the other region leaders.

Turning to the figures, FIG. **1**A illustrates an architecture of hives used in one embodiment. Shown are multiple hives **100-101**. A hive **100-101** is a logical grouping of one or more hive engines (e.g., computers or other computing devices) networked together to perform processing resources to one or more hive clients **110**. For example, hive **100** includes multiple hive engines **105-106** connected over a network (or any communication mechanism) **107**.

In one embodiment, a hive is a decentralized network of commodity hardware working cooperatively to provide vast computing power. A hive typically provides high-availability, high-scalability, low-maintenance, and predictable-time computations to applications (e.g., those corresponding to processing jobs of clients) executed in the hive. Each hive engine in the hive is typically capable to individually deploy and execute hive applications. When placed on the same network, hive engines seek each other out to pool resources and to add availability and scalability.

FIG. **1**B illustrates a computing platform used for a hive engine for implementing request handlers, process handlers, and/or other processes of a hive as used in one embodiment (or also used for simulating the operation of one or more elements of a hive in one embodiment). As shown, hive engine **120** is configured to execute request handlers, process handler, and other hive processes, and to communicate with clients and other hive engines as discussed herein.

In one embodiment, hive engine **120** includes a processing element **121**, memory **122**, storage devices **123**, communications/network interface **124**, and possibly resources/interfaces (i.e., to communicate to other resources) which may be required for a particular hive application (e.g., specialized hardware, databases, I/O devices, or any other device, etc.) Elements **121-125** are typically coupled via one or more communications mechanisms **129** (shown as a bus for illustrative purposes). Various embodiments of hive engine **120** may include more or less elements. The operation of hive engine **120** is typically controlled by processing element **121** using memory **122** and storage devices **123** to perform one or more hive processes, hive tasks, or other hive operations according to the invention. Memory **122** is one type of computer-readable medium, and typically comprises random access memory (RAM), read only memory (ROM), flash memory, integrated circuits, and/or other memory components. Memory **122** typically stores computer-executable instructions to be executed by processing element **121** and/or data which is manipulated by processing element **121** for implementing functionality in accordance with the invention. Storage devices **123** are another type of computer-readable medium, and typically comprise solid state storage media, disk drives, diskettes, networked services, tape drives, and other storage devices. Storage devices **123** typically store computer-executable instructions to be executed by processing element **121** and/or data which is manipulated by processing element **121** for implementing functionality in accordance with the invention.

In one embodiment, hive engine **120** is used as a simulation engine **120** to simulate one or more hive engines, and/or one or more hive processes, tasks, or other hive functions, such as, but not limited to those disclosed herein, especially the operations, methods, steps and communication of messages illustrated by the block and flow diagrams and messages sequence charts. Hive simulator engine **120** typically is used to simulate the performance and availability of hive application fabrics. The simulator allows dynamic simulation of any environment using simple text directives or a graphical user interface. For example, hive simulator engine **120** can be used to determine the hive performance using particular computing hardware by specifying such things as the computer type, instantiation parameters, and connection fabric, which is used by hive simulator engine **120** to produce a representation of the performance of a corresponding hive. In one embodiment, multiple hive simulator engines **120** are used, such as a unique three-level, two-dimensional mode connection fabric that allows hive simulator engines **120** to transmit requests uni-directionally or bi-directionally and to access other hive simulator engines **120** for subset processing while processing a request. Thus, one or more hive simulator engines **120** allow for modeling at the software level, hardware level, or both levels. Additionally, a hive simulator engine **120** is typically able to transmit requests through a simulated network or real hive network, such as hive **100** (FIG. **1**A).

FIG. **2**A illustrates a hierarchy of a hive, request regions, territories, and processing regions as used in one embodiment. As shown, hive **200** is logically divided into one or more request regions **205** (although most hives use only one request regions), territories **210** and **216**, with multiple processing regions **211-212** and **217-218**.

The use of territories **210** and **216** provides a mechanism for associating a physical location or quality of a corresponding hive engine which can be used, for example, in determining which responding request or process handlers to select via a volunteer pattern. When defined based on physical location, if performance is the major issue, then it is typically advantageous (but not required) to process all requests within the same territory. If reliability is the major issue, then it is typically advantageous (but not required) store state recover information in another territory.

FIG. **2**B illustrates an interaction of a client, request handlers, and process handlers of one embodiment. Client **220** generates a service request **221** to request handlers **222**, such as via a request region multicast message, one or more messages, a broadcast message, or other communication mechanisms. Those request handlers **222** that are available to process the request return responses **223** to client **220**, typically via a unicast message directly to client **220** which includes a communications port to use should the sending request handler be selected by client **220**. Client **220** selects, optionally based on territory considerations, typically one (but possibly more) of the responding request handlers, and communicates processing job **224** to the selected request handler **225**.

In response, selected request handler **225** generates a processing request **226** to process handlers **227**, such a via one or more processing region multicast messages or other communication mechanisms. Those process handlers **227** that are available to process the request return responses **228** to selected request handler **225**, typically via a unicast message directly to selected request handler **225** which includes a communications port to use should the sending request handler be selected by selected request handler **225**. Selected request handler **225** selects, optionally based on territory considerations, typically one (but possibly more) of the responding process handlers, and communicates processing job with state information **229** to the selected process handler **230**. Inclusion of the state information is emphasized in regards to processing job with state information **229** because the processing job might be ran from the beginning or initialization state, or from an intermittent position or state, such as might happen in response to an error or timeout condition.

In response, selected process handler **230** proceeds to execute the process flow (or any other specified application), and at defined points in the process flow, updates selected request handler **225** with updated/progressive state information **237**. Typically based on the process flow, selected process handler **230** will sequentially (although one embodiment allows for multiple tasks or sub-processes to be executed in parallel) cause the tasks or processing requests to be performed within the same hive engine or by other hive engines.

In one embodiment, selected process handler **230** selects a hive engine to perform a particular task using a volunteer pattern. For example, selected process handler **230** sends a multicast task request **231** to task handlers typically within the processing region (although one embodiment, sends task requests **231** to hive engines in one or more processing and/or request regions). Those task handlers **232** able to perform the corresponding task send a response message **233** to selected process handler **230**, which selects, possibly based on territory, hive engine (e.g., itself as less overhead is incurred to perform the task within the same hive engine) or other considerations, one of the responding task handlers **232**. Selected process handler **230** then initiates the task and communicates state information via message **234** to the selected task handler **235**, which performs the task and returns state information **236** to selected process handler **230**. If there are more tasks to perform, selected process handler **230** typically then repeats this process such that tasks within a process flow or application may or may not be performed by different hive engines. Upon completion of the application/process flow, selected process handler **230** forwards the final state information (e.g., the result) **237** to selected request handler **225**, which in turn, forwards the result and/or other information **238** to client **220**.

In one embodiment, selected process handler **230** performs tasks itself or causes tasks to be performed within the hive engine in which it resides (and thus selected task handler **235** is within this hive engine, and one embodiment does not send task request message **231** or it is sent internally within the hive engine.) In one embodiment, selected task handler **235** is a separate process or thread running in the same hive engine as selected process handler **230**. Upon completion of the application/process flow, selected process handler **230** forwards the final state information (e.g., the result) **237** to selected

request handler **225**, which in turn, forwards the result and/or other information **238** to client **220**.

FIG. **2**C illustrates multicast addresses **240** used in one embodiment. As shown, multicasts addresses **240** includes: a multicast request region address **241** using which a client typically sends a service request message, a processing region leader intercommunication multicast address **242** used for processing region leaders to communicate among themselves, a processing region active region indications multicast address **243** which is typically used to periodically send-out messages by region leaders to indicate which processing regions are currently active, and multiple processing region multicasts addresses **244**, one typically for each processing region of the hive. Of course, different sets or configurations of multicast addresses or even different communications mechanisms may be used in one embodiment within the scope and spirit of the invention.

FIG. **2**D illustrates the flow of messages among components of one embodiment. Client **250** sends a multicast hive service request message **256** into the request region **251** of the hive. Request handlers available for performing the application corresponding to request **256** respond with UDP messages **257** to client **250**, which selects selected request handler **252**, one of the responding request handlers. In one embodiment, this selection is performed based on territory or other considerations, or even on a random basis. Client **250** then communicates the processing job in a message **258** over a TCP connection to the selected request handler **252**.

In response and using a similar volunteer pattern, selected request handler **252** multicasts a processing request message **259** to a selected processing region **253**, and receives UDP response messages **260** from available processing engines to service the request (e.g., perform the processing job). Selected request handler **252** selects selected process handler **254**, one of the responding request handlers. In one embodiment, this selection is performed based on territory or other considerations, or even on a random basis. Selected request handler **252** then forwards the processing job with state information in message **261** to selected process handler **254**, which returns an acknowledgement message **262**. In response, selected request handler **252** sends an acknowledgement message **263** to client **250** (e.g., so that it knows that the processing is about to be performed.)

Selected process handler **254** then causes the processing job to be executed, typically by performing tasks within the same hive engine if possible for optimization reasons, or by sending out one or more tasks (possibly using a volunteer pattern) to other hive engines. Thus, selected process handler **254** optionally sends a multicast task request message **264** typically within its own processing region (i.e., selected processing region **253**) (and/or optionally to one or more other processing or request regions), and receives responses **265** indicating available task handlers for processing the corresponding task. Task request message **264** typically includes an indication of the type or name of the task or task processing to be performed so that task handlers/hive engines can use this information to determine whether they can perform the task, and if not, they typically do not send a response message **265** (as it is less overhead than sending a response message indicating the corresponding task handler/hive engine cannot perform the task.) Note, in one embodiment, a task handler within the same hive engine as selected process handler **254** sends a response message **265**.

Whether a task handler to perform the first task is explicitly or implicitly determined, selected process handler initiates a first task **266**, which is performed by one of one or more individual task threads **255** (which may be the same or different task threads on the same or different hive engines), which upon completion (whether naturally or because of an error or timeout condition), returns state information **272** to selected process handler **254**, which in turn updates selected request handler **252** via progressive state message **273**. (Note, if there was only one task, then completion/state message **276** would have been sent in response to completion of the task.) This may continue for multiple tasks as indicated by optional MCAST task request and response messages **268**-**269** and task-n initiation **270** and state messages **272**. When processing of the application/process flow is completed as determined by selected process handler **254** in response to state messages from the individual task threads **255**, selected process handler **254** forwards a completion and result state information **276** to selected process handler **252**, which forwards a result message **277** to client **250**. In response, client **250** sends an acknowledgement message **278** to confirm receipt of the result (indicating error recovery operations do not need to be performed), and an acknowledgement message **279** is forwarded to selected process handler **254**, and processing of the processing job is complete.

FIG. **2**E illustrates an interaction of a client, request handlers, process handlers and possibly tasks of one embodiment. Many of the processes and much of the flow of information is the same as illustrated in FIG. **2**B and described herein, and thus will not be repeated. FIG. **2**E is used to emphasize and explicitly illustrate that different embodiments may implement features differently, and to emphasize that a process flow may specify tasks or even other process flows to be performed or the same process flow to be performed recursively.

For example, as shown, selected process handler **230** of FIG. **2**B is replaced with selected process handler **280** in FIG. **2**E. Selected process handler **280**, in response to being assigned to execute the clients processing job by receiving processing job with state information message **229**, proceeds to execute the corresponding application/process flow, which may optionally include performing a volunteer pattern using processing or task request messages **281** and response messages **283** to/from one or more task or process handlers **282**. In response to the volunteer operation or directly in response to receiving the processing job with state information message **229**, selected process handler **280** will sequentially (although one embodiment allows for multiple tasks or subprocesses to be executed in parallel) perform itself or send out tasks or processing requests to corresponding selected task or process handlers **290**, in which case task or processing job with state information messages **284** are typically sent and results or state information messages **296** are typically received. The number of levels used in performing a processing job is unbounded as indicated in FIG. **2**E.

FIG. **3** is a flow diagram of a client process used in one embodiment. Processing begins with process block **300**, and proceeds to process block **302**, wherein an application, data, and hive to process these is identified. Next, in process block **304**, a multicast service request message indicating application is sent into the request layer of the selected hive. In process block **306**, responses are received from the hive (if no responses are received, processing returns to process block **302** or **304** in one embodiment). Next, in process block **308**, a request handler is selected based on the responses, and a communications connection is established to the selected request handler in process block **310**. Next, in process block **312**, the processing job is submitted to the selected request handler and a global unique identifier (GUID) is included so that the client and hive can uniquely identify the particular processing job. As determined in process block **314**, if an

acknowledgement message is not received from the hive indicating the job is being processed within a timeframe, then processing returns to process block **304**.

Otherwise, if results are received from the hive within the requisite timeframe as determined in process block **320**, then an acknowledgement message is returned to the hive in process block **322**, and processing is complete as indicated by process block **324**. Otherwise, as determined in process block **330**, if the client determines it wishes to perform a recover operation, then in process block **332**, a multicast recovery request message specifying the GUID is sent to the request layer of the hive, and processing returns to process block **320** to await the recovery results. Otherwise, as determined in process block **340**, if the client determines to again request the job be performed, then processing returns to process block **304**. Otherwise, local error processing is optionally performed in process block **342**, and processing is complete as indicated by process block **344**.

FIGS. **4**A-C are flow diagrams of request hander processes used in one embodiment. FIG. **4**A illustrates a process used in one embodiment for responding to service requests of clients. Processing begins with process block **400**, and proceeds to process block **402**, wherein a multicast port is opened for receiving service request messages. As determined in process blocks **404** and **406**, until a service request is received and the request handler is available to handle the request, processing returns to process block **404**. Otherwise, the request handler responds in process block **408** by sending a response message to the requesting client, with the response message typically identifying a port to use and the GUID of the received service request. As determined in process block **410**, if the service request corresponds to a recovery request, then in process block **412**, a recovery thread is initialized (such as that corresponding to the flow diagram of FIG. **4**C) or the recovery operation is directly performed. Otherwise, in process block **414**, a selected request handler thread is initialized (such as that corresponding to the flow diagram of FIG. **4**B) or the request is handled directly. Processing returns to process block **404** to respond to more requests.

FIG. **4**B illustrates a flow diagram of a process used by a selected request handler in one embodiment. Processing begins with process block **430**, and loops between process blocks **432** and **434** until a job is received (and then processing proceeds to process block **440**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **436**.

After a processing job has been received (e.g., this process has been selected by the client to handle the request), a state data structure is initialized in process block **440**. Then, in process block **442**, a multicast processing request message is sent into one of the processing layers of the hive. As determined in process block **444**, if no responses are received within a requisite timeframe, then a no processing handler response message is returned to the client in process block **445**, and processing is complete as indicated by process block **436**.

Otherwise, in process block **446**, a particular process handler is selected. In one embodiment, this selection is performed based on territories (e.g., a process handler in a different territory than the selected request handler), other considerations or even on a random basis. In process block **448**, a communications connection is established if necessary to the selected process handler, and the state information and data for the client processing request is sent (which may correspond to the initial state of the data received from the client or to an intermediate state of processing the client job request).

As determined in process block **450**, if an error or timeout condition is detected, processing returns to process block **442**. Otherwise, as determined in process block **452**, until a state update message is received, processing returns to process block **450**. As determined in process block **454**, if the received state is not the finished or completed state, then in process block **456**, the state data structure is updated, and processing returns to process block **450**. Otherwise, processing has been completed, and in process block **458**, the result is communicated to the client; in process block **460**, the communications connection is closed; and processing is complete as indicated by process block **462**.

FIG. **4**C illustrates a flow diagram of a process used by a selected request handler performing error recovery in one embodiment. Processing begins with process block **470**, and loops between process blocks **472** and **474** until a job is received (and then processing proceeds to process block **478**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **476**.

After a processing job has been received (e.g., this process has been selected by the client to perform the recover processing), in process block **478**, a multicast recovery request message specifying the GUID of the job being recovered is sent into one or more of the recovery modules of the hive. As determined in process block **480**, if no responses are received within a requisite timeframe, then a no recover response message is returned to the client in process block **481**, and processing is complete as indicated by process block **476**.

Otherwise, in process block **482**, a particular recovery handler is selected, possibly based on territory considerations—such as a recovery handler in a different territory then this selected request handler. In process block **484**, a communications connection is established if necessary to the selected recovery handler thread, and a recovery request is sent, typically including the GUID or other indication of the job to be recovered.

As determined in process block **486**, if an error or timeout condition is detected, processing returns to process block **478**. Otherwise, the recovered information is received as indicated by process block **488**. In process block **490**, the information is typically communicated to the client, or if this communication fails, it is saved to the recovery system. In one embodiment, the partially completed state, errors and/or other indications are stored to a local storage mechanism (e.g., some computer-readable medium) to be made available for use by a recovery process. In one embodiment, more significant process handling is performed, or the error communicating the error to another process, thread or hive engine for handling. The communications connection is then closed in process block **492**, and processing is complete as indicated by process block **494**.

FIGS. **5**A-B are flow diagrams of process hander processes used in one embodiment. FIG. **5**A illustrates a process used in one embodiment for responding to service requests of request handlers. Processing begins with process block **500**, and proceeds to process block **502**, wherein a multicast port is opened for receiving processing request messages. As determined in process blocks **504** and **506**, until a processing request is received and the process handler is available to handle the request, processing returns to process block **504**. Otherwise, the process handler responds in process block **508** by sending a response message to the requesting request handler, with the response message typically identifying a port to use and possibly the GUID corresponding to the received processing request. The processing request is received in process block **510**. Next, in process block **512**, a selected process handler thread is initialized (such as that corresponding to the flow diagram of FIG. **5**B) or the processing request is handled directly. Processing returns to process block **504** to respond to more requests.

FIG. **5**B illustrates a flow diagram of a process used by a selected process handler in one embodiment. Processing begins with process block **520**, and loops between process blocks **522** and **524** until a job is received (and then processing proceeds to process block **530**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **526**.

After a processing job has been received (e.g., this process has been selected by a selected request handler (or possibly other process handler) to handle the request), a state data structure is initialized in process block **530**. In process block **532**, the processing requirements of the next statement(s) within the process flow corresponding to the received job are identified. As determined in process block **534**, if a sub-process is to be spawned (e.g., the process flow specifies a process flow to be executed), then in process block **536**, the current state is pushed on to a state stack and the state is initialized to that of the new process flow, the selected request handler is updated in process block **538**, and processing returns to process block **532** to process the new process flow.

Otherwise, as determined in process block **540**, if the task handler is not already known (e.g., an optimization to perform the task on the same hive engine) such as it is not guaranteed to be performed locally, the task is a "limited task" in that it can only be performed by a subset of the task handlers or the processing of the task is made available to other hive engines (e.g., for performance or load balancing etc.), then in process block **542** the task handler to perform the task is identified. One embodiment identifies the task handler by sending a multicast task request messages, receives the responses, and selects, based on territory, load or other considerations, a task handler to perform the task.

Limited tasks provide a mechanism for identifying hive engines that have special hardware or other resources. Task handlers only on the hive engines with the specialized hardware or other resources possibly required to perform the task will be enabled to perform the corresponding task and thus these enabled task handlers will be the ones to respond to a task request for the corresponding task. Additionally, limited tasks provide a mechanism to limit the number of task handlers or hive engines allowed to access a particular resource by restricting the number and/or location of task handlers allowed to perform a task that accesses the particular resource. Thus, limited tasks may be useful to limit the rate or number of accesses to a particular resource (e.g., database engine, a storage device, a printer, etc.)

In process block **544**, a task is initiated to perform the next operation identified in the current process flow with the current state information and characteristics (e.g., timeout, number of retries, etc.) on the identified, selected, or already known task handler. As determined in process block **546**, after completion of the processing requirements of the processing statement(s), if the finish state has not been reached, then the state data structure is updated with the task result in process block **548**, the selected request handler is updated with the current state information in process block **549**, and processing returns to process block **532**.

Otherwise, processing is completed of the current process flow as determined in process block **546**, and if the current process flow is a sub-process (e.g., spawned process flow) (as determined in process block **550**), then in process block **552**, the state is popped from the state stack, and processing proceeds to process block **548**. Otherwise, in process block **554**, the result/state information is communicated to the selected request hander. As determined in process block **555**, if an error has been detected, then error processing is performed in process block **556**. In process block **558**, the communications connection is closed, and processing is complete as indicated by process block **559**. Note, in some embodiments, communications connections are not established and disconnected each time, but rather a same communications channel is used more than once.

FIG. **5**C illustrates a flow diagram of a task handler performed by a hive engine in one embodiment. Processing begins with process block **580**. As determined in process blocks **581** and **583**, until a task request is received and the task handler is available to handle the request, processing returns to process block **581**. Otherwise, the task handler responds in process block **584** by sending a response message to the requesting process (typically a process handler), with the response message typically identifying a port to use and the GUID of the received task request. As determined in process block **585**, if the task is actually received (e.g., this task handler was selected by the process handler sending the task request), then in process block **586**, the task is performed or at least attempted to be performed and resultant state information (e.g., completed state, partially completed state, errors and/or other indications) sent to the requesting process handler. Processing returns to process block **581**. Note, in one embodiment, multiple processes illustrated in process block **5**C or some variant thereof are performed simultaneously by a hive engine for responding to multiple task requests and/or performing tasks in parallel.

FIG. **5**D illustrates a flow diagram of a recovery processing performed by a hive engine in one embodiment. Processing begins with process block **590**, and loops between process blocks **591** and **592** until a recovery job is received (and then processing proceeds to process block **594**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **593**. In process block **594**, the recovery is retrieved from local storage and is communicated to the selected request hander. As determined in process block **595**, if an error has been detected, then error processing is performed in process block **595**. In process block **598**, the communications connection is closed, and processing is complete as indicated by process block **599**.

In one embodiment, a hive application is a collection of process flows that carry out specific sets of tasks. Applications can share process flows. An application definition file (XML descriptor file) typically describes the application, and the application definition file typically consists of the following: application name, process flow names, task names and module file names, support files, and/or configuration file names.

FIG. **6**A illustrates an example definition file **600** of an application for use in one embodiment. As show, application definition file **600** specifies a set of corresponding process flows **601**, tasks **602**, support files **603**, and configuration files **604**.

FIG. **6**B illustrates a definition of an process flow **620** "doProcessOne" used in one embodiment. Shown are four process flow statements **621-624**, each specifying its beginning state, tasks to be performed, and next state depending on the outcome of the statements execution.

FIG. **6**C illustrates a process used in one embodiment for executing a process flow or processing job, such as that illustrated in FIG. **6**B. Note, in one embodiment, the process illustrated in FIG. **5**B is used to execute a process flow or processing job. In one embodiment, a combination of the processes illustrated in FIGS. **5**B and **6**C or another process is used to execute a process flow or processing job.

Turning to FIG. **6**C, processing begins with process block **650**, and proceeds to process block **652**, wherein the current state is set to the START state. Next, in process block **654**, the task associated with the current state is attempted to be performed. As determined in process block **656**, if the task timed-out before completion, then as determined in process block **658**, if the task should be retried (e.g., the number of retries specified in the process flow or a default value has not been exhausted), processing returns to process block **656**. Otherwise, in process block **660**, the current state is updated to that corresponding to the task's completion status (e.g., complete, non-complete, not-attempted, etc.). As determined in process block **662**, if an error occurred (e.g., an invalid next state or other error condition), then an error indication is returned to the selected request handler in process block **664**, and processing is complete as indicated by process block **666**. Otherwise, if the next state is the FINISH state (as determined in process block **670**), then the result and possibly a final set of state information is sent to the selected request handler in process block **672**, and processing is complete as indicated by process block **672**. Otherwise, in process block **674**, the selected request handler is updated with current state information, such as, but not limited to (nor required to include) the current state name, intermediate results, variable values, etc. Processing then returns to process block **654**.

One embodiment of a hive uses a logical hierarchy of hive engines for delegation of performing administrative and/or other hive related tasks. In one embodiment, each hive engine participates in the processing region hierarchy as a region member with one hive engine in each processing region being a region leader, and there one overall senior region leader for the hive. For example, shown in FIG. **7**A are multiple processing regions **700-701**, having an overall senior region leader **703** (denoted senior leader/region leader/region member as it performs all functions) residing in processing region **700**, a region leader/region member **707** in processing region **701**, region members **704-705** in processing region **700**, and region members **708-709** in processing region **701**.

FIGS. **7**B-**7**C are flow diagrams of processes used in one embodiment to establish and maintain this hierarchical relationship among distributed processes or systems, such as among hive engines. The generic terms of heartbeat leader and heartbeat member are used in describing this process, because it can be used in many different applications for establishing and maintaining a hierarchical relationship in a set of dynamic and autonomous processes and systems. For example, in one embodiment, the processes illustrated in FIGS. **7**B-C are used to establish and maintain which hive engine in a region is the region leader, and between region leaders for establishing which hive engine is the senior region leader.

Processing of the heartbeat leader flow diagram illustrated in FIG. **7**B begins with process block **720**, and proceeds to process block **722** wherein a multicast heartbeat request message is sent on the multicast address belonging to the group in which the hierarchical relationship is being established and maintained. In process block **724**, the responses are received. As determined in process block **725**, if the process is senior over those from which a response was received, then it remains the leader or senior process, and optionally in process block **726**, piggybacked information (e.g., number of regions, number of members in each region, etc.) is processed and possibly actions taken or initiated in response. As indicated by process block **727**, the process delays or waits a certain period of time before repeating this process, and then processing returns to process block **722**. Otherwise, in process block **728**, the process demotes itself from being the leader or senior process (such as by initiating or switching to performing actions consistent with being a region member if not already performing the functions of a region member), and processing is complete as indicated by process block **729**.

Processing of the heartbeat member flow diagram illustrated in FIG. **7**C begins with process block **740**, and proceeds to process block **742**, wherein the process watches for and identifies heartbeat request messages during a predetermined timeframe. As determined in process block **744**, if a no heartbeat request is received, then in process block **745**, the process promotes itself to being the heartbeat leader, and processing returns to process block **742**. Otherwise, if this process is senior to a process sending a heartbeat request message as determined in process block **748**, then processing proceeds to process block **745** to promotes itself. Otherwise, in process block **749**, a heartbeat response message is sent to the sender of the received heartbeat request message, and optionally other information is included in the heartbeat response message. Processing then returns to process block **742**. Note, determining seniority can be performed in numerous manners and mechanisms, such as that based on some physical or logical value associated with a hive engine (e.g., one of its network addresses, its serial number, etc.)

FIG. **8**A illustrates some of the functions performed by a senior processing region leader in one embodiment. Processing begins with process block **800**, and proceeds to process block **802**, wherein a heartbeat request is sent to all region leaders, typically by sending a multicast packet to the processing region leader intercommunication multicast address **242** (FIG. **2**C) and piggybacked information is collected from received responses with this information typically including, but not limited to the number of processing regions, number of processing handlers, number of request handlers, limited task information, etc. As determined in process block **804**, if the number of request handlers needs to be adjusted (e.g., there are too few or too many), then in process block **806**, a region leader is selected and directed to start or stop a request handler. Next, as determined in process block **808**, if the number of processing regions needs to be adjusted (e.g., there are too few or too many), then in process block **810**, a region leader is selected and directed to disband or spit a region. Next, as determined in process block **812**, if the number of task handlers that can perform a particular task (i.e., a "limited task" as typically and by default, all tasks can be performed by all task handlers) needs to be adjusted (e.g., there are too few or too many), then in process block **814**, a region leader is selected and directed to adjust the number of task handlers within its region which can perform the particular limited task. Next, as determined in process block **816**, if some other action needs to be performed, then in process block **818**, the action is performed or a region leader is instructed to perform the action. Next, processing usually waits or delays for a predetermined or dynamic amount of time as indicated by process block **819**, before processing returns to process block **802**.

FIG. **8**B illustrates some of the functions performed by a region leader in one embodiment. Processing begins with process block **830**, and proceeds to process block **832**, wherein a heartbeat request is sent to all region member, typically by sending a multicast packet to the processing region multicast address **244** (FIG. **2**C), and piggybacked information is collected from received responses with this information typically including, but not limited to the number of processing handlers, number of request handlers, etc.; or possibly instructions are received from the senior region leader. As determined in process block **834**, if the number of request handlers needs to be adjusted (e.g., there are too few or too many), then in process block **836**, a process handler is selected and directed to start or stop a request handler. Next, as determined in process block **838**, if the number of processing regions needs to be adjusted (e.g., there are too few or too many), then in process block **840**, an instruction to disband or spit the region is issued. Next, as determined in process block **842**, if the number of task handlers permitted to perform a particular limited task needs to be adjusted (e.g., there are too few or too many), then in process block **844**, an instruction is provided (directly, indirectly such as via a request or process handler, or based on a volunteer pattern) to a particular task handler to permit or deny it from performing the particular limited task. Next, as determined in process block **846**, if some other action needs to be performed, then in process block **848**, the action is performed or a process handler is instructed to perform the action. Next, processing usually waits or delays for a predetermined or dynamic amount of time as indicated by process block **849**, before processing returns to process block **832**.

FIG. **8**C illustrates the splitting of a region as performed in one embodiment. Region leader **860** sends a multicast message **871** requesting a volunteer to head the new region to region members **861**, some of which typically return a positive response message **872**. Region leader **860** then identifies a selected region member **862** to head the new processing region, and sends an appointment message **873** to selected region member **862**. In response, selected region member **862** creates a new processing region as indicated by reference number **874**, typically including identifying an unused processing region multicast address **244** (FIG. **2**C) as it monitored the traffic or processing region active indication messages sent to processing region active region indications multicast address **243**. Then, selected region member **862** multicasts a volunteer message **875** to processing regions in the old (and still used) processing region and typically receives one or more responses **876**. Selected region member **862** then selects a certain number, typically half of the number of process handlers in the old processing region, of responding process handlers, and notifies them to switch to the new processing region via move instruction **877**, and they in turn, send a confirmation message **878** to selected region member **862**.

FIG. **9** illustrates a process used in one embodiment for initializing a hive engine. Processing begins with process block **900**, and proceeds to process block **902**, wherein a hive version request and hive join multicast message is sent typically to all region leaders. As determined in process block **904**, if no responses are received, then in process block **912**, a new processing region is formed, and request hander and region leader processes are initiated. Next, in process block **914**, process handler, recovery module, and region member processes are initiated, and startup processing is completed as indicated by process block **916**. Otherwise, as determined in process block **906**, if a hive software update is available, then, in process block **908**, one of the responders is selected, the updates are acquired, and the software (e.g., hive software, operating system, etc.) is updated. In process block **910**, the hive engine joins the smallest or possibly one of the smaller processing regions, possibly with this selection being determined by identified territories, and processing proceeds to process block **914**.

In one embodiment, the hive is updated by a client with special administrative privileges. This administrative client sends a request to the senior region leader of the hive. The senior region leader opens a TCP connection and sends the administration client the connection information. The administration client sends the new application to the senior region leader. When the senior region leader receives an update, the senior region leader multicasts the update command to all the hive members. The senior region leader sends multicast message containing the name of the file that is being updated, the new version, and the total number of packets each hive member should receive. The senior region leader then multicasts the data packets, each packet typically includes the file id, the packet number, and data. If a hive member does not receive a packet, that hive member sends a request to the senior region leader for the missing packet. The senior region leader resends, multicasts, the missing packet. The hive members store the update in a staging area until they receive the activation command. To activate an update, the administration client sends the activation command to the senior region leader. The senior region leader multicasts the activate command to the hive members. The hive members remove the old application or files and moves the update from the staging area to the production area. To update the hive software or operating system, the senior region leader distributes the updates and restarts volunteers in a rolling fashion. When the hive service manager detects a new version of itself, the service manager forks the process and restarts with a new version. Also, the senior region leader can send other update commands. An active message indicates that the corresponding application, patch, or OS that should be running on the hive. A deactivated messages indicates that the corresponding application, patch, or OS should not be running on the hive and should remain installed on hive members. A remove message indicates that the corresponding application, patch, or OS was once installed on the Hive and any instances found on Hive members should be removed. This allows hive engines to be updated and also to move back to previous releases.

In view of the many possible embodiments to which the principles of our invention may be applied, it will be appreciated that the embodiments and aspects thereof described herein with respect to the drawings/figures are only illustrative and should not be taken as limiting the scope of the invention. For example and as would be apparent to one skilled in the art, many of the process block operations can be re-ordered to be performed before, after, or substantially concurrent with other operations. Also, many different forms of data structures could be used in various embodiments. The invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

What is claimed is:

1. A system for processing information, the system comprising:

a plurality of networked computers for processing a processing job in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job;

the request handler configured to (1) receive a service request for the processing job, and (2) communicate data representative of the processing job to a process handler;

the process handler to which the processing job data was communicated being configured to (1) receive the communicated processing job data, and (2) analyze the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers;

the task handlers configured to (1) perform the processing tasks of the processing job in accordance with the determined sequence, and (2) generate updated state information in response to the performed processing tasks; and

wherein the request handler is further configured to (1) maintain state information for the processing job based on the updated state information, (2) determine whether a fault exists, and (3) in response to a determination that a fault exists, initiate a recovery procedure based on the maintained state information for the processing job.

2. The system of claim 1 wherein the request handler is further configured to initiate the recovery procedure by communicating the maintained state information for the processing job to another of the process handlers for the another process handler to pick up the processing job in accordance with the maintained state information.

3. The system of claim 1 wherein the request handler is further configured to (1) receive an error code regarding the processing job, and (2) determine that a fault exists in response to the received error code.

4. The system of claim 3 wherein the request handler is further configured to receive the error code from the process handler to which the processing job data was communicated.

5. The system of claim 1 wherein the request handler is further configured to (1) determine whether a timeout condition exists, and (2) in response to a determination that a timeout condition exists, determine that a fault exists.

6. The system of claim 1 wherein the updated state information generated by the task handlers comprises updated task state information, wherein the task handlers are further configured to communicate the updated task state information to the process handler to which the processing job data was communicated, and wherein the process handler to which the processing job data was communicated is further configured to (1) generate updated state information for the processing job based on the updated task state information, and (2) communicate the updated state information to the request handler.

7. The system of claim 6 wherein the process handler to which the processing job was communicated is further configured to (1) determine whether a fault exists, and (2) in response to a determination that a fault exists, initiate a recovery procedure with a task handler based on the updated state information for the processing job that it generated.

8. The system of claim 1 wherein the process flow further comprises a timeout parameter associated with a processing task to control when a fault is determined to exist.

9. The system of claim 1 wherein a secondary process handler is configured to (1) redundantly store state information for the processing job, and (2) in response to recovery procedure initiation, pick up the processing job in accordance with its stored state information.

10. The system of claim 1 wherein at least one of the networked computers has a process handler and a task handler resident therein.

11. A method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising:

receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job;

the request handler (1) receiving a service request for the processing job, and (2) communicating data representative of the processing job to a process handler;

the process handler to which the processing job data was communicated (1) receiving the communicated processing job data, and (2) analyzing the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers;

the task handlers (1) performing the processing tasks of the processing job in accordance with the determined sequence, and (2) generating updated state information in response to the performed processing tasks; and

the request handler (1) maintaining state information for the processing job based on the updated state information, (2) determining whether a fault exists, and (3) in response to a determination that a fault exists, initiating a recovery procedure based on the maintained state information for the processing job.

12. The method of claim **11** wherein the initiating step comprises the request handler communicating the maintained state information for the processing job to another of the process handlers for the another process handler to pick up the processing job in accordance with the maintained state information.

13. The method of claim **11** further comprising the request handler receiving an error code regarding the processing job, and wherein the fault determining step comprises the request handler determining that a fault exists in response to the received error code.

14. The method of claim **13** wherein the error code receiving step comprises the request handler receiving the error code from the process handler to which the processing job data was communicated.

15. The method of claim **11** further comprising the request handler determining whether a timeout condition exists, and wherein the fault determining step comprises the request handler determining that a fault exists in response to a determination that a timeout condition exists.

16. The method of claim **11** wherein the updated state information generated by the task handlers comprises updated task state information, the method further comprising (1) the task handlers communicating the updated task state information to the process handler to which the processing job data was communicated, and (2) the process handler to which the processing job data was communicated (i) generating updated state information for the processing job based on the updated task state information, and (ii) communicating the updated state information to the request handler.

17. The method of claim **16** further comprising the process handler to which the processing job was communicated (1) determining whether a fault exists, and (2) in response to a determination that a fault exists, initiating a recovery procedure with a task handler based on the updated state information for the processing job that it generated.

18. The method of claim **11** wherein the process flow further comprises a timeout parameter associated with a processing task to control when a fault is determined to exist.

19. The method of claim **11** further comprising a secondary process handler (1) redundantly storing state information for the processing job, and (2) in response to recovery procedure initiation, picking up the processing job in accordance with its stored state information.

20. The method of claim **11** wherein at least one of the networked computers has a process handler and a task handler resident therein.

21. A method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising:

the request handler receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job;

a process handler coordinating an execution of the processing tasks by a plurality of the task handlers;

the task handlers (1) executing the processing tasks, and (2) generating updated state information for the processing job in response to the executing step;

as the processing tasks are executed by the task handlers, redundantly storing updated state information across a plurality of different processes;

determining whether a failure has occurred; and

in response to a determination that a failure has occurred, (1) retrieving a copy of the redundantly stored state information, and (2) resuming the processing job in accordance with the retrieved state information.

22. The method of claim **21** wherein the redundantly storing step comprises, as the processing tasks are executed by the task handlers, redundantly storing updated state information across a plurality of the networked computers.

23. The method of claim **22** wherein the networked computers on which the state information is redundantly stored include networked computers that are located in physically different locations.

24. The method of claim **22** wherein the failure determining step comprises the request handler determining whether a failure has occurred, and wherein the resuming step comprises another process handler resuming the processing job in accordance with the retrieved state information.

25. The method of claim **24** wherein the retrieving step comprises the request handler performing the retrieving step.

26. The method of claim **21** wherein at least one of the networked computers has a process handler and a task handler resident therein.

27. A system for processing information, the system comprising:

a plurality of networked computers for processing a processing job in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job;

the request handler configured to (1) receive a service request for the processing job, and (2) communicate data representative of the processing job to a process handler;

the process handler to which the processing job data was communicated being configured to (1) receive the communicated processing job data, and (2) analyze the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers;

the task handlers configured to (1) perform the processing tasks of the processing job in accordance with the determined sequence, and (2) generate updated state information in response to the performed processing tasks; and

wherein the process handler to which the processing job data was communicated is further configured to (1) maintain state information for the processing job based on the updated state information, (2) determine whether a fault exists, and (3) in response to a determination that a fault exists, initiate a recovery procedure based on the maintained state information for the processing job.

28. A method for processing information via a plurality of networked computers, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the method comprising:

receiving a service request for the processing job, the processing job comprising a process flow, the process flow including (1) a plurality of processing tasks and (2) state information relating to the processing job;

the request handler (1) receiving a service request for the processing job, and (2) communicating data representative of the processing job to a process handler;

the process handler to which the processing job data was communicated (1) receiving the communicated processing job data, and (2) analyzing the processing job data and state information to determine a sequence of processing tasks to be performed by the task handlers;

the task handlers (1) performing the processing tasks of the processing job in accordance with the determined sequence, and (2) generating updated state information in response to the performed processing tasks; and

the process handler to which the processing job data was communicated (1) maintaining state information for the processing job based on the updated state information, (2) determining whether a fault exists, and (3) in response to a determination that a fault exists, initiating a recovery procedure based on the maintained state information for the processing job.

29. A system for processing information, the system comprising:

a plurality of networked computers for processing a plurality of processing jobs in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow;

the request handler configured to (1) receive a plurality of service requests for the processing jobs, (2) store state information for the processing jobs, and (3) select a plurality of process handlers from among the process handlers for servicing the processing jobs;

the selected process handlers configured to (1) analyze the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identify a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determine that the processing job corresponding to the process flow with no remaining processing tasks has been completed; and

the task handlers configured to perform the identified processing tasks to generate a plurality of task results, the task results causing an update to the state information for the processing job.

30. The system of claim **29** wherein the task handlers are further configured to volunteer for performing the identified tasks based on their being able to perform the identified tasks.

31. The system of claim **30** wherein the selected process handlers are further configured to communicate a plurality of task requests for the identified processing tasks to a plurality of the task handlers, wherein the tasks handlers to which the task requests were communicated are further configured to volunteer for performing the identified tasks based on their being able to perform the identified tasks, and wherein the selected process handlers are further configured to select the task handlers for performing the identified tasks from among the task handlers that volunteered for same.

32. The system of claim **29** wherein the selected process handlers are further configured to communicate a plurality of messages to the task handlers to initiate the task handlers to perform the identified processing tasks.

33. The system of claim **29** wherein the task handlers are further configured to communicate updated state information for the processing jobs following completion of their identified processing tasks to the selected process handlers.

34. The system of claim **33** wherein the selected process handlers are further configured to communicate updated state information for the processing jobs to the request handler.

35. The system of claim **29** wherein the process handlers are further configured to volunteer for servicing the processing jobs based on their availabilities for servicing the processing jobs.

36. The system of claim **35** wherein the request handler is further configured to communicate a plurality of processing requests for the processing jobs to a plurality of the process handlers, wherein the process handlers to which the processing requests were communicated are further configured to volunteer for servicing the processing jobs corresponding to the processing requests based on their availabilities for servicing the processing jobs corresponding to the processing requests, and wherein the request handler is further configured to select the process handlers for servicing the processing jobs corresponding to the processing requests from among the process handlers that volunteered for same.

37. The system of claim **29** wherein the request handler is further configured to (1) receive at least one of the service requests from a client computer, and (2) communicate a processing result for the processing job corresponding to the at least one service request to the client computer.

38. The system of claim **37** further comprising the client computer, the client computer in communication via a network with the networked computer on which the request handler is resident.

39. The system of claim **29** wherein the request handler is resident on a first of the networked computers, wherein at least one of the process handlers is resident on a second of the networked computers, and wherein at least one of the task handlers is resident on a third of the networked computers.

40. The system of claim **29** wherein the request handler and at least one of the process handlers are resident on the same one of the networked computers.

41. The system of claim **29** wherein the request handler is resident on a different one of the networked computers than the networked computers on which the process handlers and task handlers are resident.

42. The system of claim **41** wherein the process handlers are resident on different ones of the networked computers than the networked computers on which the task handlers are resident.

43. The system of claim **29** wherein at least one of the process handlers and at least one of the task handlers are resident on the same one of the networked computers, the at least one task handler being on a different thread of the networked computer than the at least one process handler.

44. The system of claim **29** wherein the logic for a process flow comprises (1) a plurality of state variables before and after each processing task of that process flow, the state variables configured to store the state information, and (2) a plurality of transitions from state to state based on results for the processing tasks of that process flow.

45. The system of claim **44** wherein the request handler, the selected process handlers, and the task handlers are configured to process the processing jobs according to a plurality of process definition files corresponding to the processing jobs, each process definition file being configured to define the process flow for a processing job.

46. The system of claim **45** wherein each process definition file further comprises a recovery procedure for a plurality of the processing tasks.

47. The system of claim **46** wherein the process definition files comprise XML process definition files.

48. The system of claim **29** wherein at least one of the process flows further includes a recovery procedure for the processing tasks of the at least one process flow.

* * * * *

US009049267B2

(12) **United States Patent**
Hinni et al.

(10) **Patent No.:** **US 9,049,267 B2**
(45) **Date of Patent:** ***Jun. 2, 2015**

(54) SYSTEM AND METHOD FOR PROCESSING INFORMATION VIA NETWORKED COMPUTERS INCLUDING REQUEST HANDLERS, PROCESS HANDLERS, AND TASK HANDLERS

(71) Applicant: **Appistry, Inc.**, St. Louis, MO (US)

(72) Inventors: **Aaron Louis Hinni**, Ballwin, MO (US); **Guerry Anderson Semones**, Marthasville, MO (US); **Michael Scott Groner**, Chesterfield, MO (US); **Roberto Raul Lozano**, Creve Coeur, MO (US)

(73) Assignee: **Appistry, Inc.**, St. Louis, MO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/176,378**

(22) Filed: **Feb. 10, 2014**

(65) **Prior Publication Data**

US 2014/0156722 A1 Jun. 5, 2014

Related U.S. Application Data

(60) Continuation of application No. 13/707,861, filed on Dec. 7, 2012, now Pat. No. 8,682,959, which is a continuation of application No. 13/491,893, filed on Jun. 8, 2012, now Pat. No. 8,341,209, which is a

(Continued)

(51) **Int. Cl.**

G06F 15/16	(2006.01)
H04L 29/06	(2006.01)

(Continued)

(52) **U.S. Cl.**
CPC ***H04L 67/42*** (2013.01); ***H04L 41/00*** (2013.01); ***H04L 67/16*** (2013.01); ***H04L 69/16*** (2013.01); ***H04L 67/10*** (2013.01); ***H04L 69/163*** (2013.01)

(58) **Field of Classification Search**
CPC H04L 67/16; H04L 67/10; H04L 41/00
USPC 709/201–203, 223, 224
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,357,632	A	10/1994	Pian et al.
5,742,762	A	4/1998	Scholl et al.

(Continued)

FOREIGN PATENT DOCUMENTS

JP	H10-222487 A	8/1998
KR	1019980051002	1/2001
WO	03/010659 A1	2/2003

OTHER PUBLICATIONS

Beranek et al., "Host Access Protocol Specification", RFC 907, Jul. 1984, 79 pages.

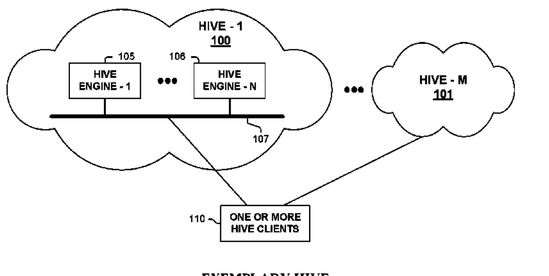
(Continued)

Primary Examiner — Mohamed Wasel
(74) *Attorney, Agent, or Firm* — Thompson Coburn LLP

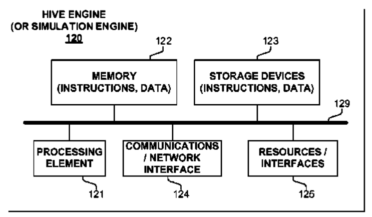
(57) **ABSTRACT**

Systems and methods for processing information via networked computers leverage request handlers, process handlers, and task handlers to provide efficient distributed processing of processing jobs. A request handler can receive service requests for processing jobs, process handlers can identify tasks to be performed in connection with the processing jobs, and task handlers can perform the identified tasks, where the request handler, the process handlers, and the task handlers can be distributed across a plurality of networked computers.

139 Claims, 25 Drawing Sheets



EXEMPLARY HIVE COMPUTING ARCHITECTURE


HIVE ENGINE (OR SIMULATION ENGINE)
120
122
123
MEMORY (INSTRUCTIONS, DATA)
STORAGE DEVICES (INSTRUCTIONS, DATA)
129
PROCESSING ELEMENT
COMMUNICATIONS / NETWORK INTERFACE
RESOURCES / INTERFACES
121
124
125

Related U.S. Application Data

continuation of application No. 13/293,527, filed on Nov. 10, 2011, now Pat. No. 8,200,746, which is a division of application No. 12/127,070, filed on May 27, 2008, now Pat. No. 8,060,552, which is a division of application No. 10/236,784, filed on Sep. 7, 2002, now Pat. No. 7,379,959.

(51) **Int. Cl.**

H04L 12/24	(2006.01)
H04L 29/08	(2006.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,748,489	A	5/1998	Beatty et al.	
5,790,789	A	8/1998	Suarez	
5,892,913	A	4/1999	Adiga et al.	
5,894,554	A	4/1999	Lowery et al.	
5,937,388	A	8/1999	Davis et al.	
6,070,190	A	5/2000	Reps et al.	
6,128,277	A	10/2000	Bruck et al.	
6,144,990	A	11/2000	Brandt et al.	
6,247,109	B1	6/2001	Kleinsorge et al.	
6,249,291	B1	6/2001	Popp et al.	
6,282,697	B1	8/2001	Fables et al.	
6,470,346	B2	10/2002	Morwood	
6,484,224	B1	11/2002	Robins et al.	
6,564,240	B2	5/2003	Waldo et al.	
6,665,701	B1	12/2003	Combs et al.	
6,766,348	B1	7/2004	Combs et al.	
7,035,933	B2	4/2006	O'Neal et al.	
7,043,225	B1 *	5/2006	Patel et al.	455/405
7,197,547	B1	3/2007	Miller et al.	
7,379,959	B2	5/2008	Hinni et al.	
8,060,552	B2	11/2011	Hinni et al.	
8,200,746	B2	6/2012	Hinni et al.	
8,341,209	B2	12/2012	Hinni et al.	
8,682,959	B2	3/2014	Hinni et al.	
2002/0023117	A1	2/2002	Bernardin et al.	
2002/0152106	A1 *	10/2002	Stoxen et al.	705/8
2006/0117212	A1	6/2006	Meyer et al.	
2006/0198386	A1	9/2006	Liu et al.	

OTHER PUBLICATIONS

Coulouris et al., "Distributed Systems Concepts and Design", 2001, pp. 515-552, Third Edition, Chapter 13, Pearson Education Limited.
Coulouris et al., "Distributed Systems Concepts and Design", 2001, pp. 553-606, Third Edition, Chapter 14, Pearson Education Limited.
Postel, "Assigned Numbers", RFC 755, May 3, 1979, 12 pages.
Veizades et al., "Service Location Protocol", RFC 2165, Jun. 1997, 72 pages.
High Throughput Computing Resource Management, Morgan Kaufmann Publishers (1997).
Internet Information Services 5.0 Technical Overview, Microsoft (2014).
Introduction—Internet Information Services Resource Guide, Microsoft (2014).
Kielmann, T., et al, "Bandwidth-efficient Collective Communication for Clustered Wide Area Systems", IEEE (2000).
Kielmann, T., et al, "MagPle: MPI's Collective Communication Operations for Clustered Wide Area Systems", ACM (1999).
Kielmann, T., et al, MPI's Reduction Operations in Clustered Wide Area Systems (1999).
Kielmann, T., et al, The Albatross Project: Parallel Application Support for Computational Grids (2000).
Kuskin, J. et al, "The Stanford FLASH Multiprocessor", IEEE (1994).
Litigation Pleadings; *Appistry, Inc.* v. *Amazon.com, Inc. and Amazon Web Services, Inc.*; Case 4:2013-cv-02547; filed Eastern District of Missouri, Dec. 20, 2013 (transferred to Case 2:2015-cv-00311 (Western District of Washington)); 2871 pages.
Litzkow, M.J., et al, "Condor—A Hunter of Idle Workstations", IEEE (1988).
Load Balancing COM+ Components, Microsoft (Mar. 1, 2002).
LSF Administrator's Guide Version 4.1, Platform Computing Corp. (Feb. 2001).
LSF Batch Administrator's Guide, 6th Ed., Platform Computing Corp. (Aug. 1998).
LSF Batch User's Guide, 6th Ed., Platform Computing Corp. (Aug. 1998).
LSF Programmer's Guide Version 3.2, Platform Computing Corp. (Aug. 1998).
McClure, S. and R. Wheeler, MOSIX: How Linux Clusters Solve Real World Problems (2000).
Microsoft Application Center 2000 Component Load Balancing Technology Overview, Microsoft (Sep. 1, 2000).
Microsoft Cluster Service, Microsoft (Jun. 9, 2001).
Minar, N. et al, "Hive: Distributed Agents for Networking Things", IEEE (Aug. 3, 1999).
Nakada, H. et al, "Design and Implementations of Ninf: towards a Global Computing Infrastructure", Elsevier Science (Oct. 2, 1998).
Nakada, H. et al, "Utilizing the Metaserver Architecture in the Ninf Global Computing System", Springer Berlin Heidelberg (1998).
Network Load Balancing, Windows 2000 Advanced Server, Microsoft (2014).
Neuman, B.C. and S. Rao, "Resource Management for Distributed Parallel Systems", IEEE (Jul. 1993).
Neuman, B.C. and S. Rao, "The Prospero Resource Manager: A scalable framework for processor allocation in distributed systems", John Wiley & Sons, Ltd. (1994).
Newburn, C.J. and J.P. Shen, Automatic Partitioning of Signal Processing Programs for Symmetric Multiprocessors, Parallel Architectures and Compilation Techniques (Oct. 1996).
Nisan, N. et al, "Globally Distributed Computation over the Internet—The POPCORN Project", IEEE (1998).
Oracle 8i Concepts, Oracle Corp. (Dec. 1999).
Oracle 9i Net Services Administrator's Guide, Oracle Corp. (Jun. 2001).
Oracle Internet Application Server 8i Overview Guide, Oracle Corp. (Jul. 2000).
Oracle WebServer 2.0, Oracle Corp. (Mar. 1996).
Petition for Inter Partes Review re USPN 8,200,746 filed by Amazon.com, Inc. and Amazon Web Services, Inc.; Dec. 22, 2014; 331 pages.
Pfaffenberger, "Script", Webster's New World Computer Dictionary—9th Edition, 3 pages.
Plaat, A. et al, "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects", IEEE (1999).
"Plaintiffs Disclosure of Asserted Claims and Preliminary Infringement Contentions"; *Appistry, Inc.* v. *Amazon.com, Inc. and Amazon Web Services, Inc.*; Case 4:2013-cv-02547; filed Eastern District of Missouri, Dec. 20, 2013 (also Case 2:2015-cv-00311 (Western District of Washington)); 314 pages.
Platform Overview, Microsoft (Jun. 9, 2001).
Rao, S., Prospero Resource Manager's 1.1 User's Guide (1996).
Regey, O. and N. Nisan, "The POPCORN Market—an Online Market for Computational Resources", ACM (1998).
Roure, D. et al, "The Evolution of the Grid", John Wiley & Sons Ltd. (2003).
Sato, M. et al, "Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure", Springer Berlin Heidelberg (1997).
Smarr, L. and C.E. Catlett, "Metacomputing", ACM (Jun. 1992).
Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual, Sun Microsystem, Inc. (Apr. 2002).
Sun Grid Engine 5.3 Manual, Sun Grid Engine, Enterprise Edition 5.3 Manual, Sun Microsystem, Inc., (Jul. 2001).
Sun ONE Grid Engine and Sun ONE Grid Engine, Enterprise Edition Reference Manual, Sun Microsystem, Inc. (Oct. 2002).
Sunderam, V.S., "PVM: A Framework for Parallel Distributed Computing", John Wiley & Sons Ltd. (Dec. 1990).
Tamayo, P., et al, "Interpreting Patterns of Gene Expression with Self-Organizing Maps: Methods and Application to Hematopoietic Differentiation", The National Academy of Sciences (1999).

(56) **References Cited**

OTHER PUBLICATIONS

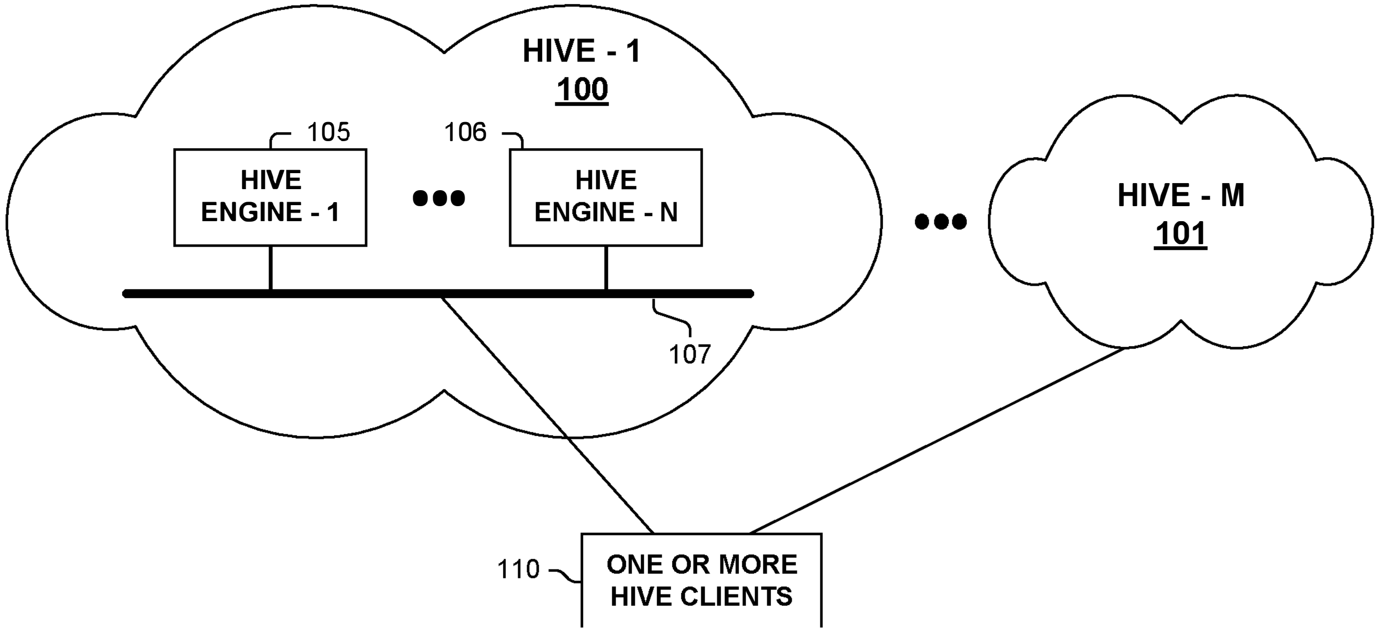
Tanenbaum, A.S. et al, "Experiences with the Amoeba Distributed ●perating System", ACM (Dec. 199●).
Tang, H. and T. Yang, "●ptimizing Threaded MPI Execution of SMP Clusters", ACM (2●●1).
Tannenbaum, T., et al, "Condor—A Distributed Job Scheduler", MIT Press (2●●2).
Tatebe, ●., et al, "Grid Datafarm Architecture for Petascale Data Intensive Computing", Asia Pacific Grid (May 23, 2●●2).
Tatebe, ●., et al, "Grid Datafarm Architecture for Petascale Data Intensive Computing", IEEE (2●●2).
Thain, D., et al, "Distributed Computing in Practice: The Condor Experience", John Wiley & Sons Ltd. (2●●4).
Thain, D., et al, "Gathering at the Well: Creating Communities for Grid I/●", ACM (2●●1).
The Buzz About Hive Computing: Putting Peer-to-Peer Computing to Work, Geneer Corp. (2●●1).
Tygar, J.D. and A. Whitten, "WWW Electronic Commerce and Java Trojan Horses", Usenix (Nov. 1996).
User's Guide, Sun Grid Engine Ch. 3 (Jul. 2●●1).
van der Aalst, W.M.P., "The Application of Petri Nets to Workflow Management", Journal of Circuits, Systems and Computers (Feb. 1998).
van Nieuwpoort, R. et al, "Wide-Area Parallel Progranuning Using the Remote Method Invocation Model", John Wiley & Sons Ltd. (1999).
van Nieuwpoort, R., et al, "Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications", ACM (2●●1).
van Steen, M. et al, "Globe: A Wide-Area Distributed System", IEEE (1999).
Vogels, W. et al, "The Design and Architecture of the Microsoft Cluster Service—A Practical Approach to High-Availability and Scalability", 199x IEEE (Jun. 23-25, 1998).
Vogels, W. et al, Scalability of the Microsoft Cluster Service (Aug. 3-5, 1998).
Web Server Clustering, Microsoft (Jun. 8, 2●●●).
Weissinger, A.K., ASP in a Nutshell, ●'Reilly & Associates, Inc. (1999).
Weissman, J. and A. Grimshaw, "A Federated Model for Scheduling in Wide-Area Systems", IEEE (Aug. 1996).
Windows 2●●● Clustering Technologies: Cluster Service Architecture, Microsoft (2●14).
Windows 2●●● Web and Application Services Technical ●verview, Microsoft (2●14).
Windows 2●●● Web Server Best Practices for High Availability, Microsoft (●ct. 1, 2●●1).
Windows Clustering Technologies—An ●verview, Microsoft (Nov. 1, 2●●1).
Aberer, K., P-Grid: A Self-●rganizing Access Structure for P2P Information Systems, Springer Berlin Heidelberg 2●●1).
Abramson, D., et al, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker", Elsevier Science B.V. (2●●2).
Abramson, D., et al, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", IEEE (2●●●).
Agrawal, D.P. et al, "Structure of a Parallelizing Complier for the B-HIVE Multicomputer", Elsevier Science B.V. (1988).
Alahakoon, D. et al, "Dynamic Self-●rganizing Maps with Controlled Growth for Knowledge Discovery", IEEE (May 2●●●).
ArcIMS 4 Architecture and Functionality, ESRI (Apr. 2●●2).
Arnold, D. et al, "Innovations of the NetSolve Grid Computing System", John Wiley & Sons, Ltd. (2●●2).
Arnold, D. et al, Users' Guide to NetSolve V1.4 (2●●1).
Bal, H., et al, "The Distributed ASCI Supercomputer Project", ACM (●ct. 2●●●).
Bal, H., et al, Parallel Computing on Wide-Area Clusters: the Albatross Project (1999).
Barak A. and ●. La'adan, The M●SIX multicomputer operating system for high performance cluster computing, Elsevier Science B.V. (1998).
Barak, A., et al, Scalable Cluster Computing with M●SIX for LINUX (1999).
Belloum, A. et al, JERA: A Scalable Web Server (1998).
Belloum, A. et al, Scalable Federations of Web Caches (Nov. 8, 1999).
Bernardt, G. et al, "A survey of load sharing in networks of workstations", The British Computer Society, The Institution of Electrical Engineers and I●P Publishing Ltd. (1993).
Breshears, C.P. and G. Fagg, "A Computation Allocation Model for Distributed Computing Under MPI_Connect", CEWES/Major Shared Resource Center/Department of Defense (Apr. 15, 1999).
Bugnion, E. et al, "Disco: Running Commodity ●perating Systems on Scalable Multiprocessors", ACM (Nov. 1997).
Buyya, R., Chapter 1—Economic-based Distributed Resource Management and Scheduling for Grid Computing (Apr. 12, 2●●2).
Buyya, R., et al, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", IEEE (May 2●●●).
Buyya, R., The Nimrod-G Grid Resource Broker and Economic Scheduling Algorithms (Apr. 12, 2●●2).
Capello, F., et al, "HiHCoHP—Toward a Realistic Communication Model for Hierarchical HyperClusters of Heterogeneous Processors", IEEE (2●●1).
Cardellini et al., "Geographic Load Balancing for Scalable Distributed Web Systems", 9 pages.
Casanova, H and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems", IEEE (1996).
Chapin, J. et al, "Hive: Fault Containment for Shared-Memory Multiprocessors", ACM (Dec. 1995).
Chapin, J., "Hive: ●perating System Fault Containment for Shared-Memory Multiprocessors", Stanford University (Jul. 1997).
Chapter 1—●verview of Internet Information Services 5.●, Microsoft (2●14).
Chapter 2—Core IIS Administration, Microsoft (2●14).
Chapter 6—Developing Web Applications, Microsoft (2●14).
Chapter 7—Data Access and Transactions, Microsoft (2●14).
Cole, G. et al, "Initial ●perating Capability for the Hypercluster Parallel-Processing Test Bed", NASA (Mar. 6-8, 1989).
Components and Web Application Architecture, Microsoft (2●14).
Czajkowski et al., "A Resource Management Architecture for Metacomputing Systems", 1998, Springer Berlin Heidelberg, 19 pages.
Czajkowski, K., et al, "A Resource Management Architecture for Metacomputing Systems", Springer Berlin Heidelberg (1998), 21 pages.
"Defendants Amazon.com Inc. and Amazon Web Services, Inc.'s Preliminary Invalidity Contentions"; *Appistry, Inc.* v. *Amazon.com, Inc. and Amazon Web Services, Inc.*; Case 4:2●13-cv-●2547; filed Eastern District of Missouri, Dec. 2●, 2●13 (also Case 2:2●15-cv-●●311 (Western District of Washington)); 461 pages.
Deploying Windows 2●●● with IIS 5.● for Dot Coms: Best Practices, Microsoft (2●14).
Distributed Queueing System—3.3.2 User Guide (Dec. 12, 2●●●).
Distributed Queueing System—3.3x (Dec. 12, 2●●●).
Dusseau, A. et al, "Effective Distributed Scheduling of Parallel Workloads", ACM (1996).
East, R., et al, "The Architecture of ArcIMS, a Distributed Internet Map Server", Springer Berlin Heidelberg (2●●1).
Epema, D.H.J., et al, "A Worldwide Flock of Condors: Load Sharing among Workstation Clusters", Esevier B.V. (1996).
Fagg, G.E. et al, "Scalable Networked Information Processing Environment (SNIPE)", Elsevier Science, B.V. (Dec. 14, 1998).
Foster I., and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of High Performance Computing Applications (Jun. 1997).
Foster, I., et al, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", IEEE (1999).
Frey, J., et al, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Kluwer Academic Publishers (Jul. 2●●2).

(56) **References Cited**

OTHER PUBLICATIONS

Gamache, R. et al, "Windows NT Clustering Service", IEEE (●ct. 1998).
Govil, K. et al, "Cellular Disco: Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors", ACM (Dec. 1999).
Grimshaw, A., et al, "The Legion Vision of a Worldwide Virtual Computer", ACM (Jan. 1997).
Grimshaw, A., et al, "Wide-Area Computing: Resource Sharing on a Large Scale", IEEE (May 1999).
Gruber, B. and K. Rossi, ●racle Webserver User's Guide, ●racle Corp. (1996).
Hariri et al., "Virtual Distributed Computing Environment", Air Force Research Laboratory Information Directorate Rome Research Site—Final Technical Report, Approved for Public Release, Mar. 2●●●, 51 pages.

* cited by examiner



EXEMPLARY HIVE
COMPUTING ARCHITECTURE

FIGURE 1A

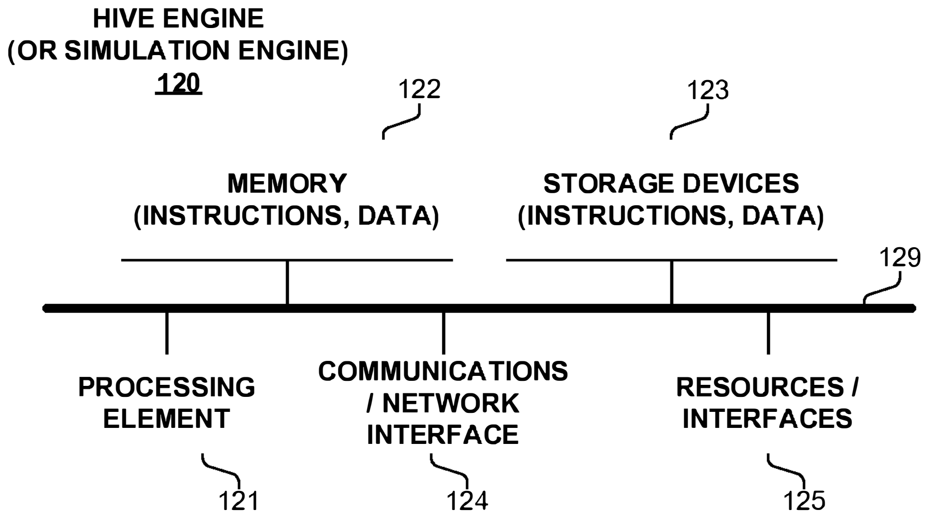

HIVE ENGINE
(OR SIMULATION ENGINE)
120
122
123
MEMORY
(INSTRUCTIONS, DATA)
STORAGE DEVICES
(INSTRUCTIONS, DATA)
129
PROCESSING
ELEMENT
COMMUNICATIONS
/ NETWORK
INTERFACE
RESOURCES /
INTERFACES
121
124
125

FIGURE 1B

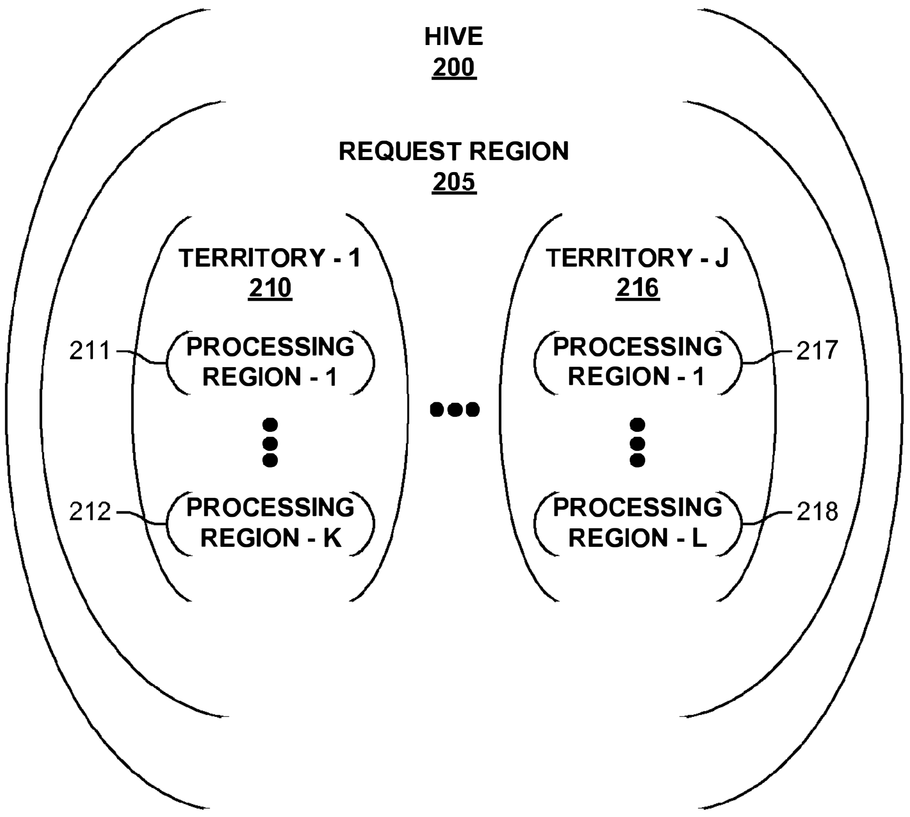

HIVE
200
REQUEST REGION
205
TERRITORY - 1
210
211
PROCESSING REGION - 1
212
PROCESSING REGION - K
TERRITORY - J
216
217
PROCESSING REGION - 1
218
PROCESSING REGION - L

FIGURE 2A

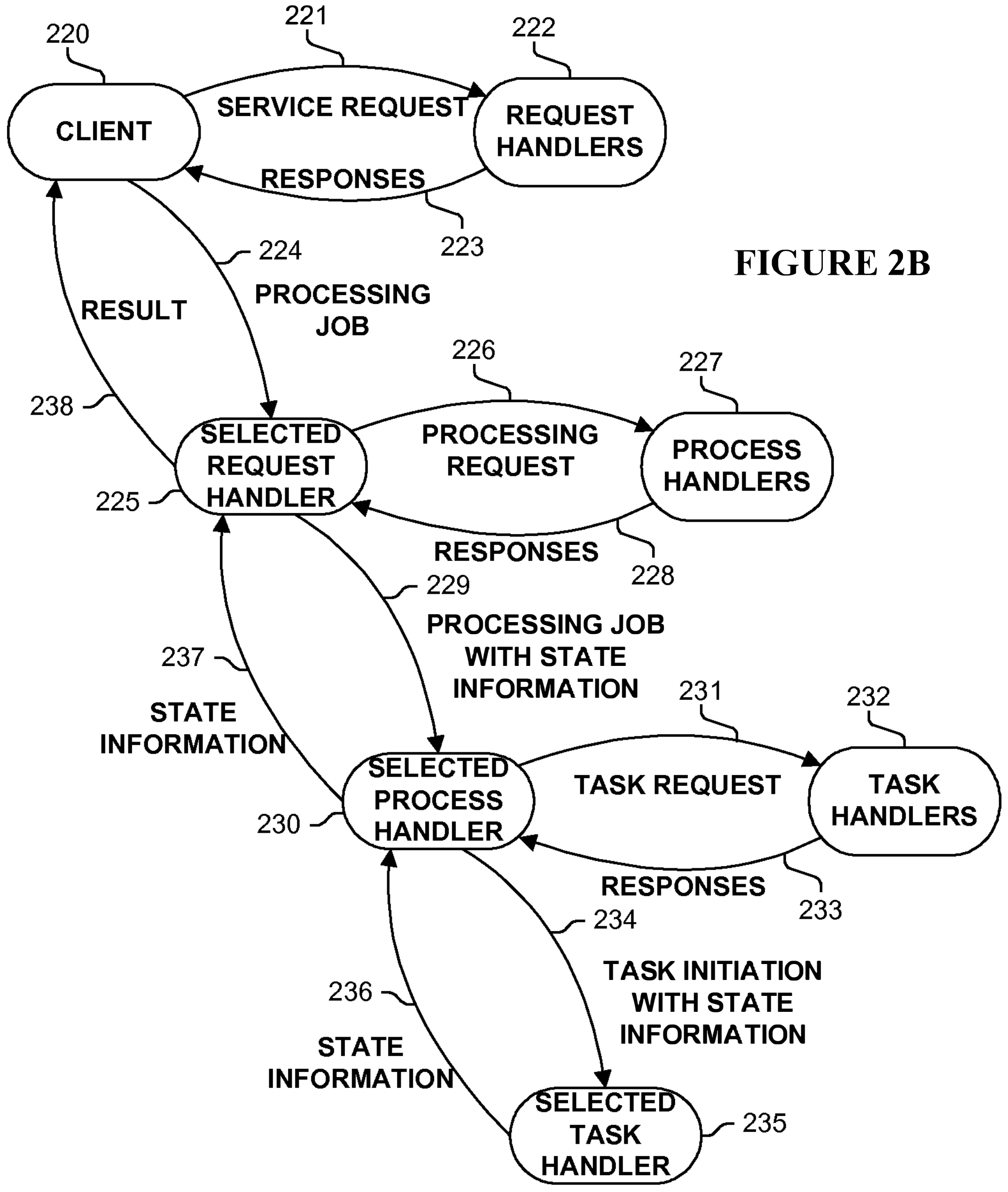

220
221
222
CLIENT
SERVICE REQUEST
REQUEST HANDLERS
RESPONSES
223
224
PROCESSING JOB
RESULT
238
226
227
SELECTED REQUEST HANDLER
PROCESSING REQUEST
PROCESS HANDLERS
225
RESPONSES
228
229
PROCESSING JOB WITH STATE INFORMATION
237
STATE INFORMATION
231
232
SELECTED PROCESS HANDLER
TASK REQUEST
TASK HANDLERS
230
RESPONSES
233
234
TASK INITIATION WITH STATE INFORMATION
236
STATE INFORMATION
SELECTED TASK HANDLER
235

FIGURE 2B

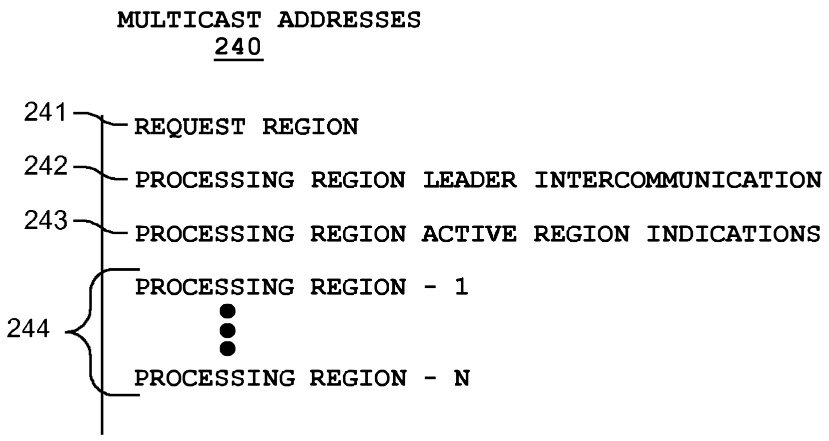
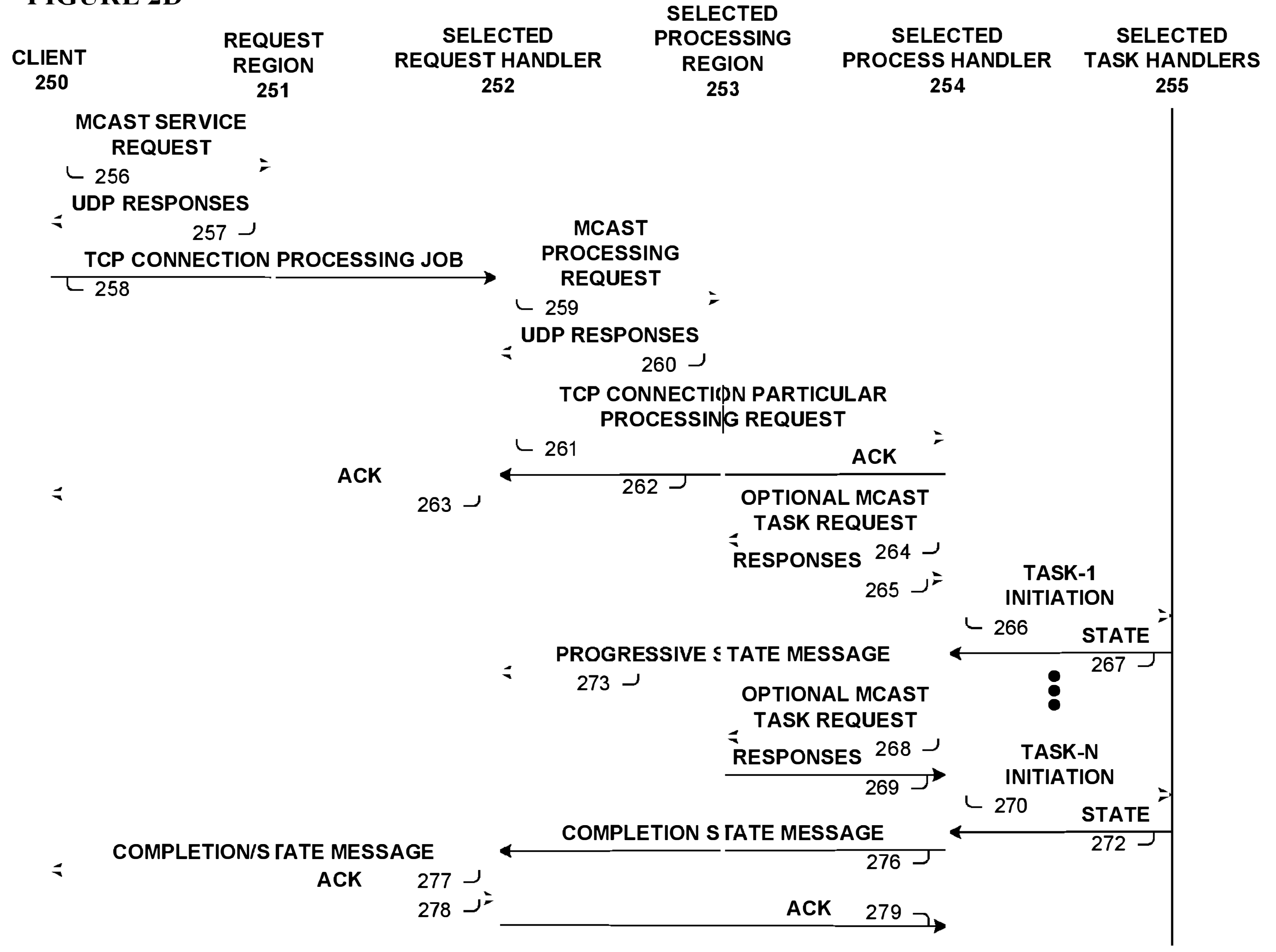

MULTICAST ADDRESSES
240
241
REQUEST REGION
242
PROCESSING REGION LEADER INTERCOMMUNICATION
243
PROCESSING REGION ACTIVE REGION INDICATIONS
244
PROCESSING REGION - 1
PROCESSING REGION - N

FIGURE 2C

FIGURE 2D


CLIENT 250
REQUEST REGION 251
SELECTED REQUEST HANDLER 252
SELECTED PROCESSING REGION 253
SELECTED PROCESS HANDLER 254
SELECTED TASK HANDLERS 255
MCAST SERVICE REQUEST
256
UDP RESPONSES
257
TCP CONNECTION PROCESSING JOB
258
MCAST PROCESSING REQUEST
259
UDP RESPONSES
260
TCP CONNECTION PARTICULAR PROCESSING REQUEST
261
ACK
262
ACK
263
OPTIONAL MCAST TASK REQUEST
RESPONSES
264
265
TASK-1 INITIATION
266
STATE
267
PROGRESSIVE STATE MESSAGE
273
OPTIONAL MCAST TASK REQUEST
RESPONSES
268
269
TASK-N INITIATION
270
STATE
272
COMPLETION STATE MESSAGE
276
COMPLETION/STATE MESSAGE
ACK
277
278
ACK
279

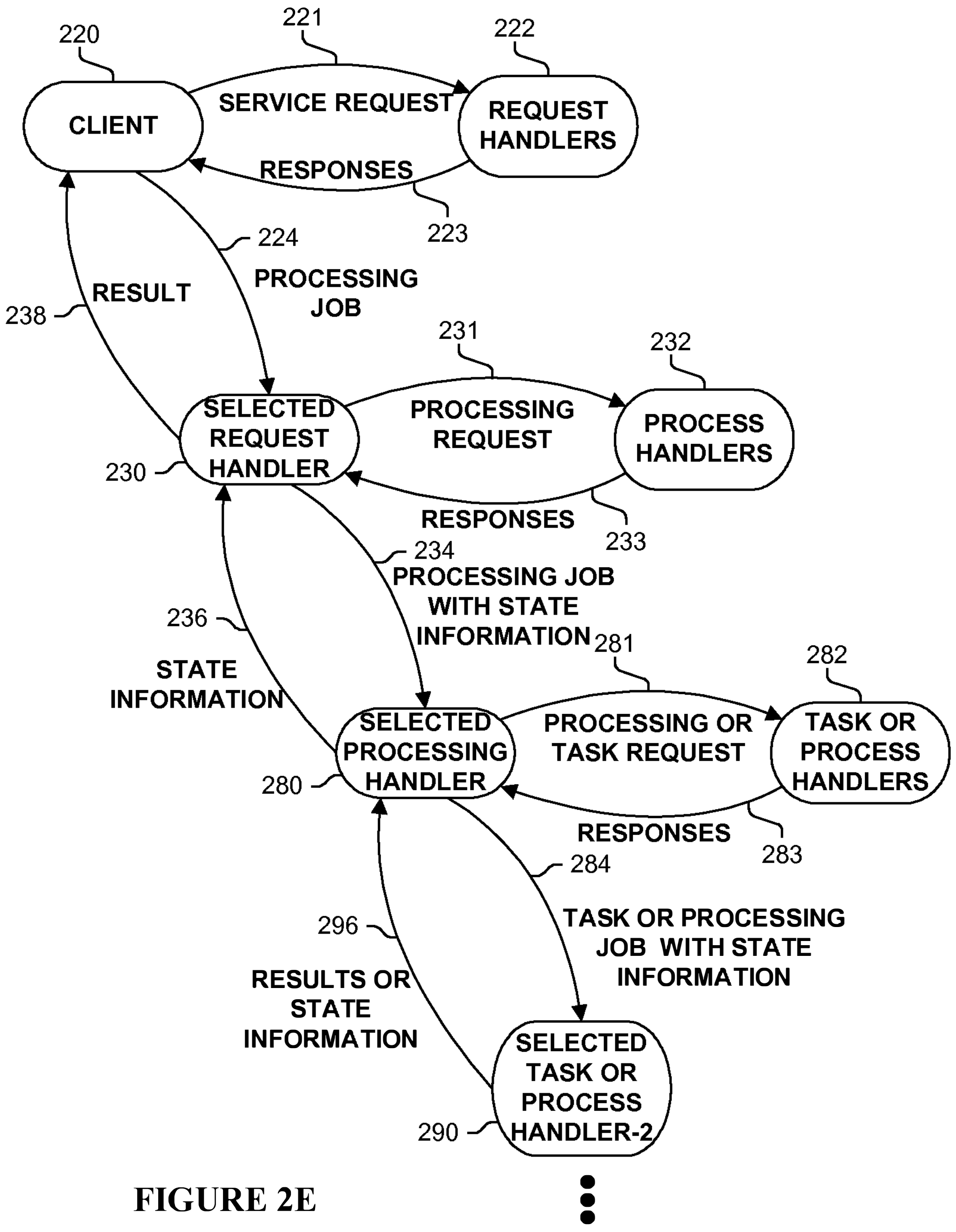
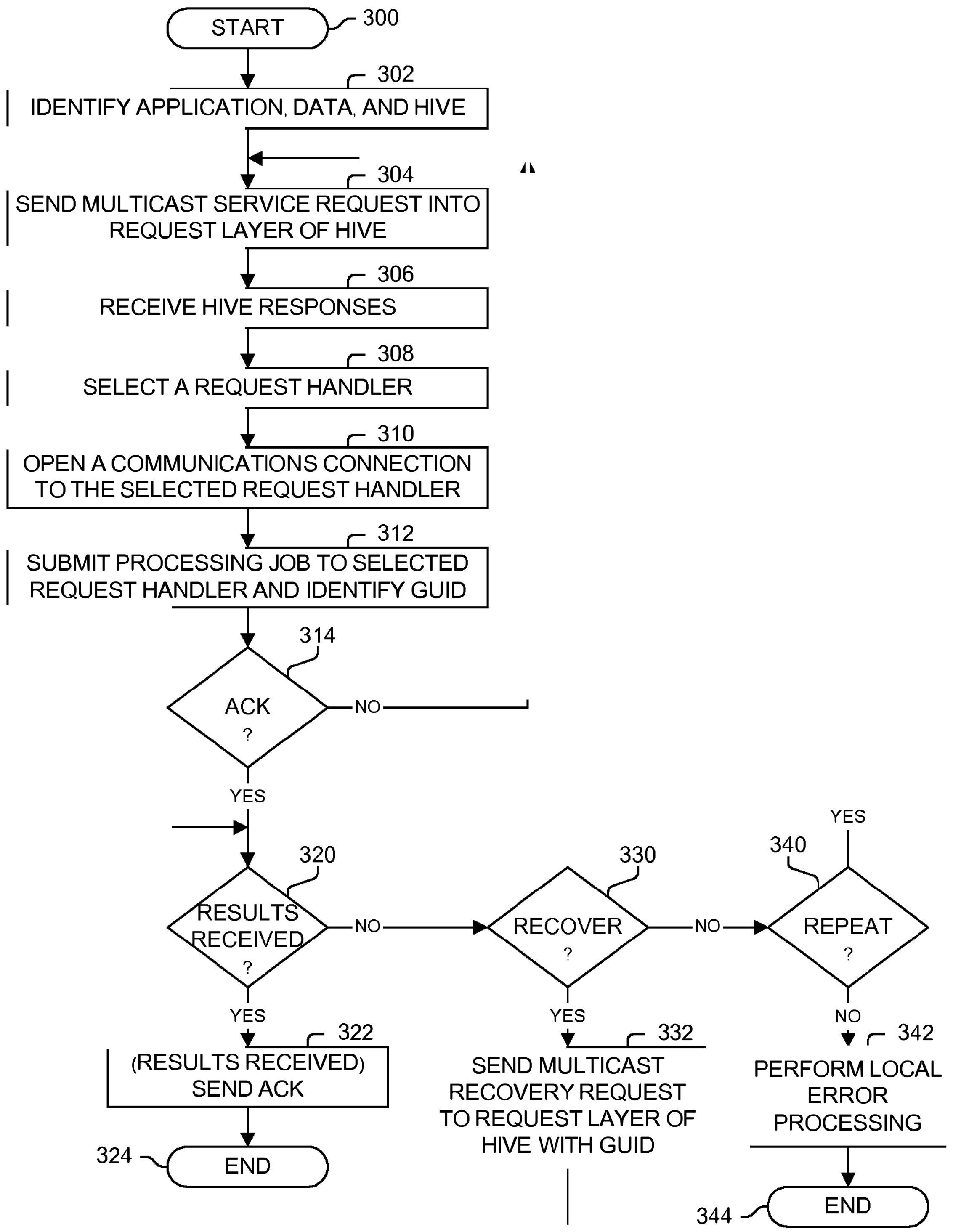
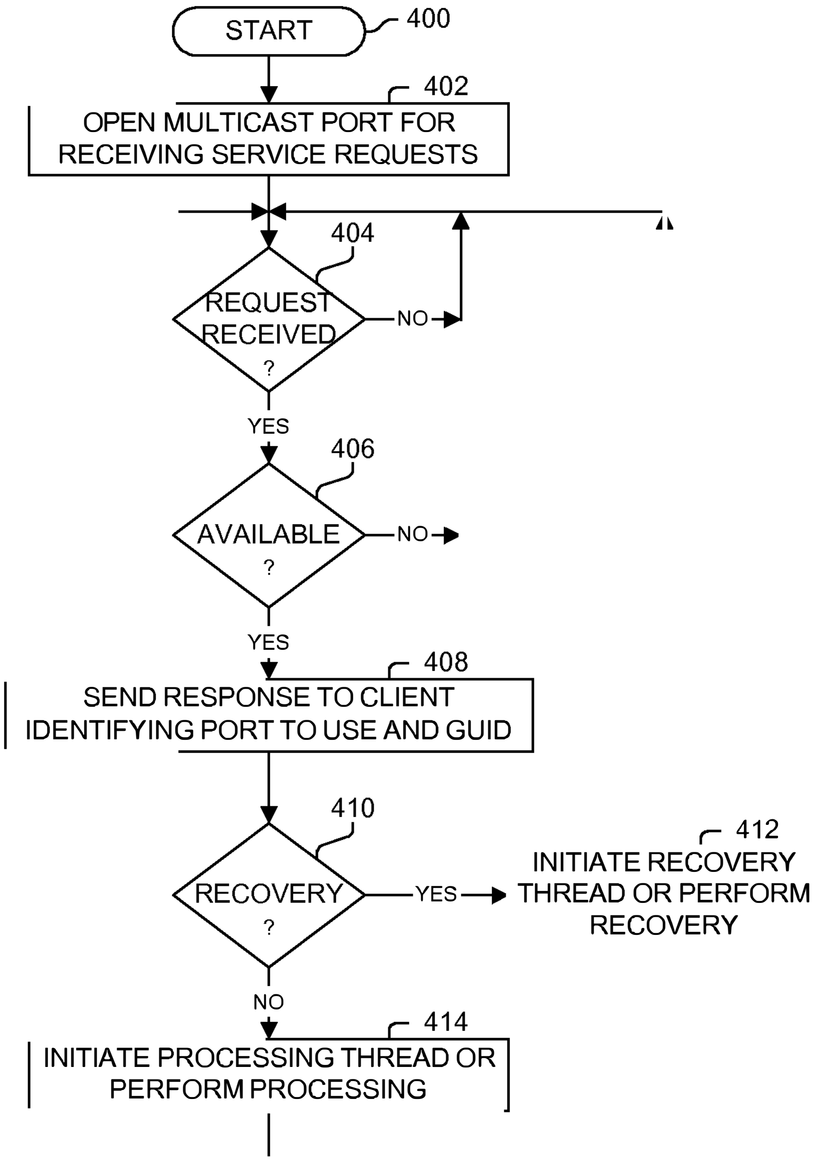

220
CLIENT
221
SERVICE REQUEST
222
REQUEST HANDLERS
RESPONSES
223
224
PROCESSING JOB
238
RESULT
231
232
SELECTED REQUEST HANDLER
PROCESSING REQUEST
PROCESS HANDLERS
230
RESPONSES
233
234
PROCESSING JOB WITH STATE INFORMATION
236
STATE INFORMATION
281
282
SELECTED PROCESSING HANDLER
PROCESSING OR TASK REQUEST
TASK OR PROCESS HANDLERS
280
RESPONSES
283
284
TASK OR PROCESSING JOB WITH STATE INFORMATION
296
RESULTS OR STATE INFORMATION
SELECTED TASK OR PROCESS HANDLER-2
290

FIGURE 2E


START
300
302
IDENTIFY APPLICATION, DATA, AND HIVE
304
SEND MULTICAST SERVICE REQUEST INTO REQUEST LAYER OF HIVE
306
RECEIVE HIVE RESPONSES
308
SELECT A REQUEST HANDLER
310
OPEN A COMMUNICATIONS CONNECTION TO THE SELECTED REQUEST HANDLER
312
SUBMIT PROCESSING JOB TO SELECTED REQUEST HANDLER AND IDENTIFY GUID
314
ACK ?
NO
YES
320
RESULTS RECEIVED ?
NO
YES
330
RECOVER ?
NO
YES
340
REPEAT ?
YES
NO
322
(RESULTS RECEIVED) SEND ACK
324
END
332
SEND MULTICAST RECOVERY REQUEST TO REQUEST LAYER OF HIVE WITH GUID
342
PERFORM LOCAL ERROR PROCESSING
344
END

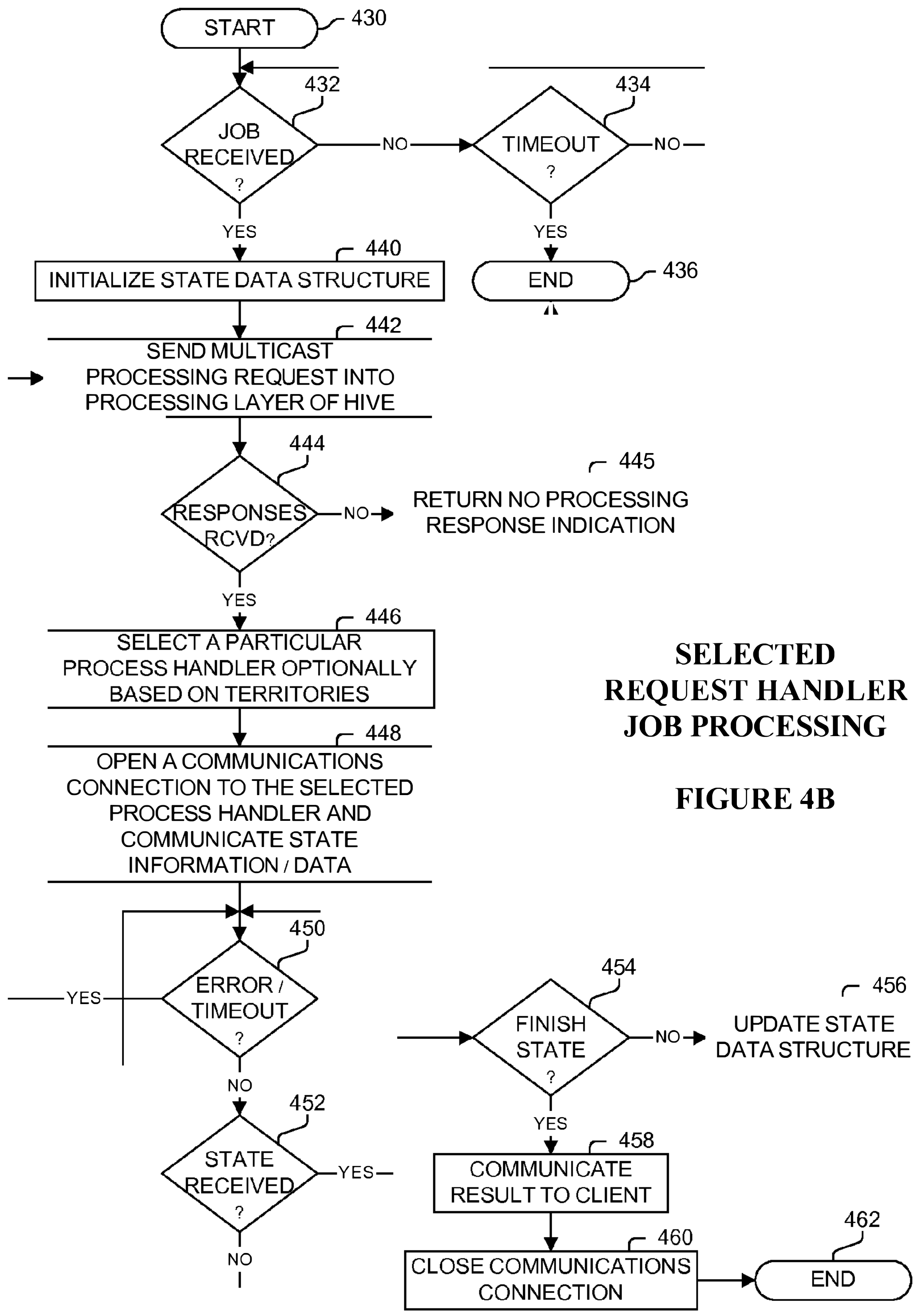
CLIENT PROCESSING

FIGURE 3


START
400
402
OPEN MULTICAST PORT FOR RECEIVING SERVICE REQUESTS
404
REQUEST RECEIVED ?
NO
YES
406
AVAILABLE ?
NO
YES
408
SEND RESPONSE TO CLIENT IDENTIFYING PORT TO USE AND GUID
410
RECOVERY ?
YES
412
INITIATE RECOVERY THREAD OR PERFORM RECOVERY
NO
414
INITIATE PROCESSING THREAD OR PERFORM PROCESSING

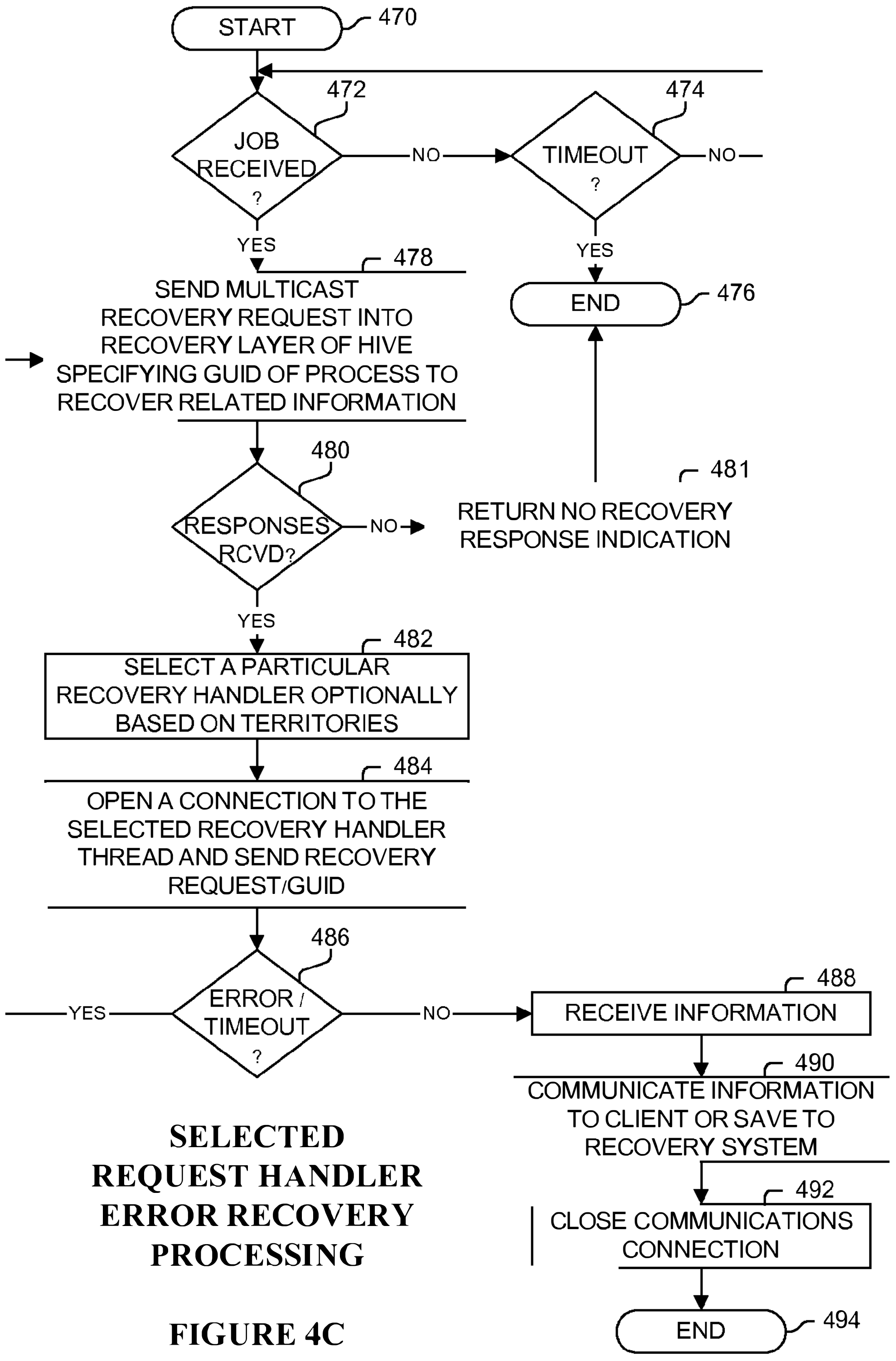
REQUEST HANDLER
RESPONSE PROCESSING

FIGURE 4A


START
430
432
JOB RECEIVED ?
NO
434
TIMEOUT ?
NO
YES
YES
440
INITIALIZE STATE DATA STRUCTURE
END
436
442
SEND MULTICAST PROCESSING REQUEST INTO PROCESSING LAYER OF HIVE
444
RESPONSES RCVD?
NO
445
RETURN NO PROCESSING RESPONSE INDICATION
YES
446
SELECT A PARTICULAR PROCESS HANDLER OPTIONALLY BASED ON TERRITORIES
448
OPEN A COMMUNICATIONS CONNECTION TO THE SELECTED PROCESS HANDLER AND COMMUNICATE STATE INFORMATION / DATA
450
YES
ERROR / TIMEOUT ?
NO
454
FINISH STATE ?
NO
456
UPDATE STATE DATA STRUCTURE
YES
452
STATE RECEIVED ?
YES
NO
458
COMMUNICATE RESULT TO CLIENT
460
CLOSE COMMUNICATIONS CONNECTION
462
END

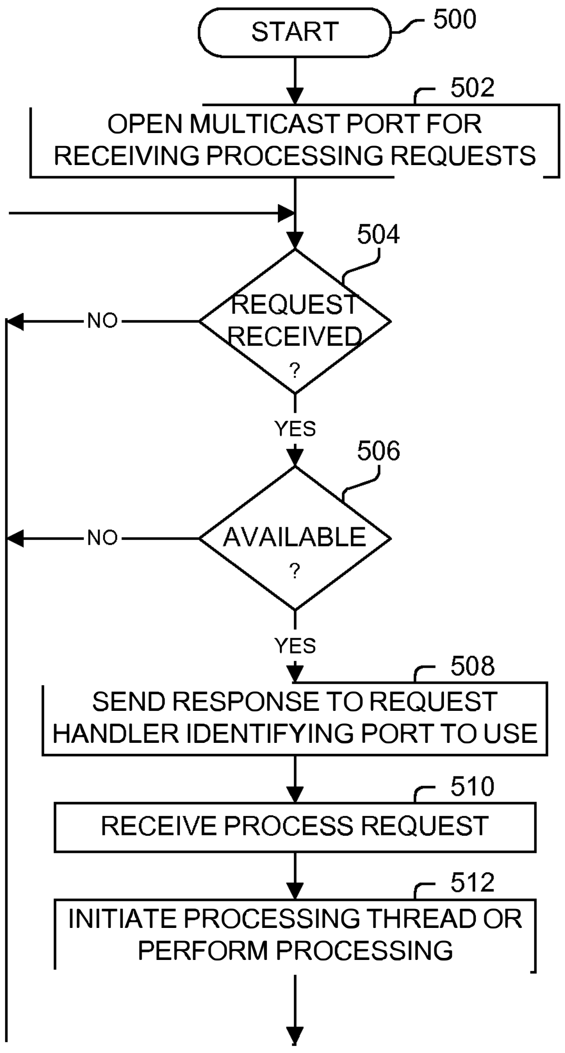
SELECTED REQUEST HANDLER JOB PROCESSING

FIGURE 4B


START
470
472
JOB RECEIVED ?
NO
474
TIMEOUT ?
NO
YES
YES
478
SEND MULTICAST RECOVERY REQUEST INTO RECOVERY LAYER OF HIVE SPECIFYING GUID OF PROCESS TO RECOVER RELATED INFORMATION
END
476
480
RESPONSES RCVD?
NO
481
RETURN NO RECOVERY RESPONSE INDICATION
YES
482
SELECT A PARTICULAR RECOVERY HANDLER OPTIONALLY BASED ON TERRITORIES
484
OPEN A CONNECTION TO THE SELECTED RECOVERY HANDLER THREAD AND SEND RECOVERY REQUEST/GUID
486
YES
ERROR / TIMEOUT ?
NO
488
RECEIVE INFORMATION
490
COMMUNICATE INFORMATION TO CLIENT OR SAVE TO RECOVERY SYSTEM
492
CLOSE COMMUNICATIONS CONNECTION
END
494

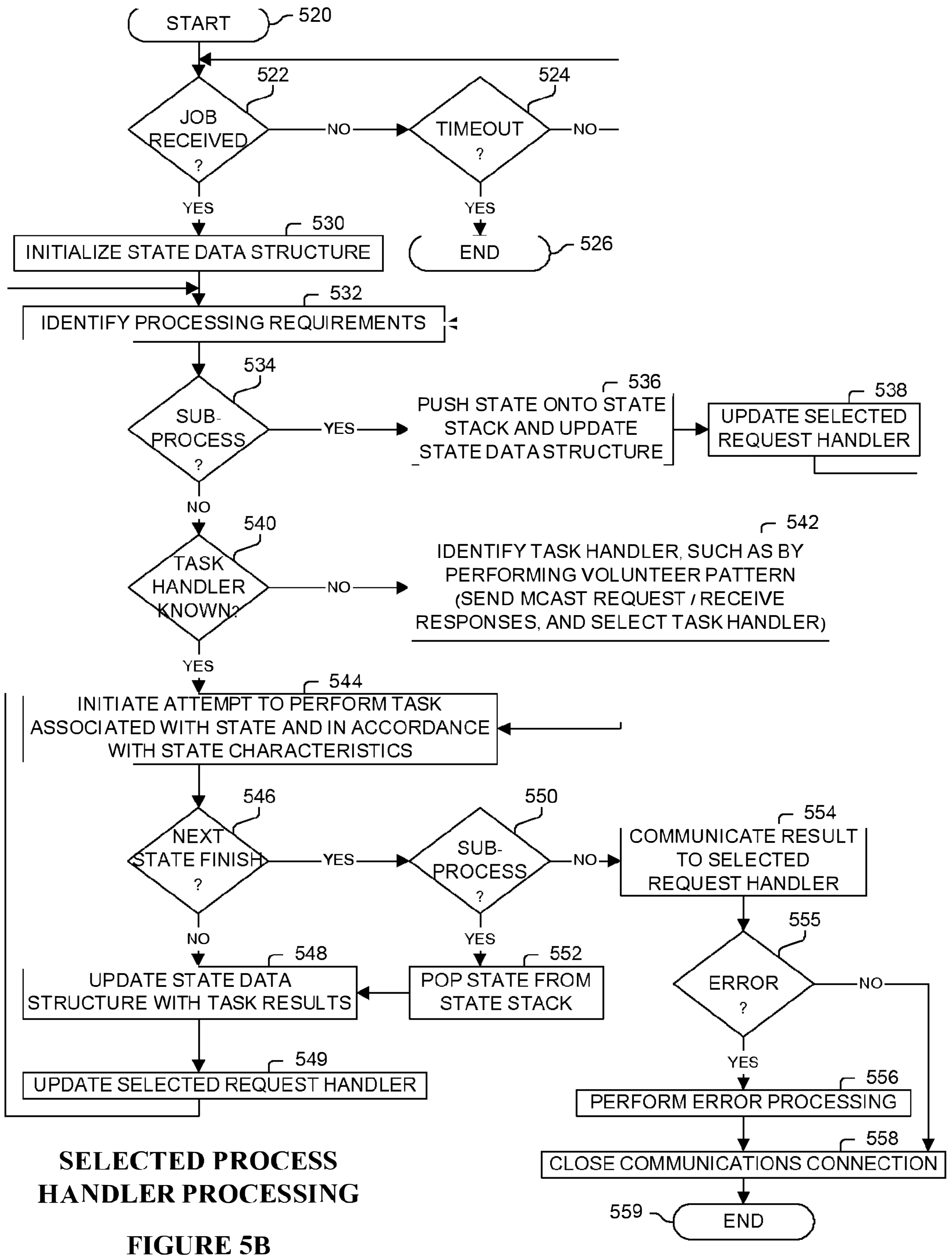
SELECTED REQUEST HANDLER ERROR RECOVERY PROCESSING

FIGURE 4C


START
500
OPEN MULTICAST PORT FOR RECEIVING PROCESSING REQUESTS
502
504
REQUEST RECEIVED ?
NO
YES
506
AVAILABLE ?
NO
YES
508
SEND RESPONSE TO REQUEST HANDLER IDENTIFYING PORT TO USE
510
RECEIVE PROCESS REQUEST
512
INITIATE PROCESSING THREAD OR PERFORM PROCESSING

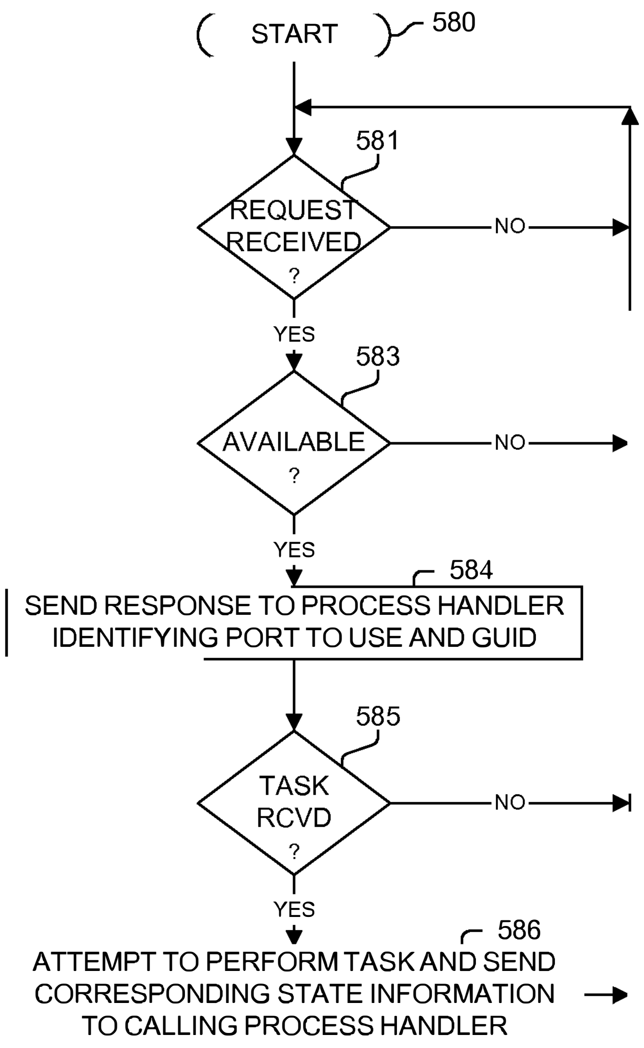
PROCESS HANDLER
RESPONSE PROCESSING

FIGURE 5A


START
520
522
JOB RECEIVED ?
NO
524
TIMEOUT ?
NO
YES
YES
530
INITIALIZE STATE DATA STRUCTURE
END
526
532
IDENTIFY PROCESSING REQUIREMENTS
534
SUB-PROCESS ?
YES
536
PUSH STATE ONTO STATE STACK AND UPDATE STATE DATA STRUCTURE
538
UPDATE SELECTED REQUEST HANDLER
NO
540
TASK HANDLER KNOWN?
NO
542
IDENTIFY TASK HANDLER, SUCH AS BY PERFORMING VOLUNTEER PATTERN (SEND MCAST REQUEST / RECEIVE RESPONSES, AND SELECT TASK HANDLER)
YES
544
INITIATE ATTEMPT TO PERFORM TASK ASSOCIATED WITH STATE AND IN ACCORDANCE WITH STATE CHARACTERISTICS
546
NEXT STATE FINISH ?
YES
550
SUB-PROCESS ?
NO
554
COMMUNICATE RESULT TO SELECTED REQUEST HANDLER
555
NO
YES
548
UPDATE STATE DATA STRUCTURE WITH TASK RESULTS
552
POP STATE FROM STATE STACK
ERROR ?
NO
YES
549
UPDATE SELECTED REQUEST HANDLER
556
PERFORM ERROR PROCESSING
558
CLOSE COMMUNICATIONS CONNECTION
559
END

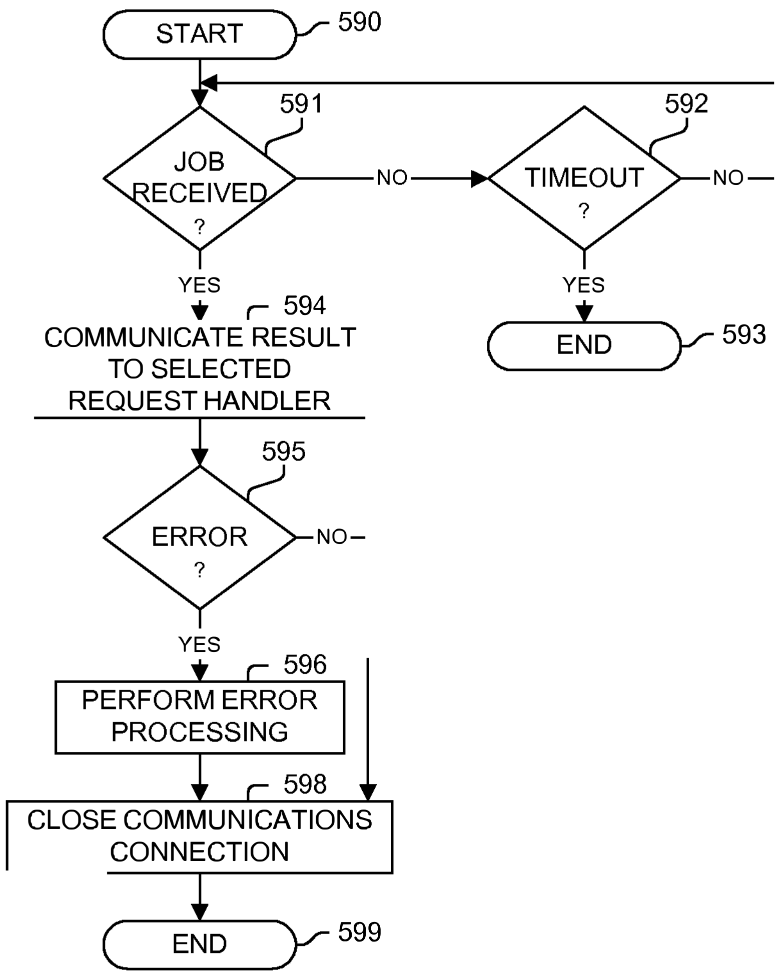
SELECTED PROCESS HANDLER PROCESSING

FIGURE 5B


START
580
581
REQUEST RECEIVED ?
NO
YES
583
AVAILABLE ?
NO
YES
584
SEND RESPONSE TO PROCESS HANDLER IDENTIFYING PORT TO USE AND GUID
585
TASK RCVD ?
NO
YES
586
ATTEMPT TO PERFORM TASK AND SEND CORRESPONDING STATE INFORMATION TO CALLING PROCESS HANDLER

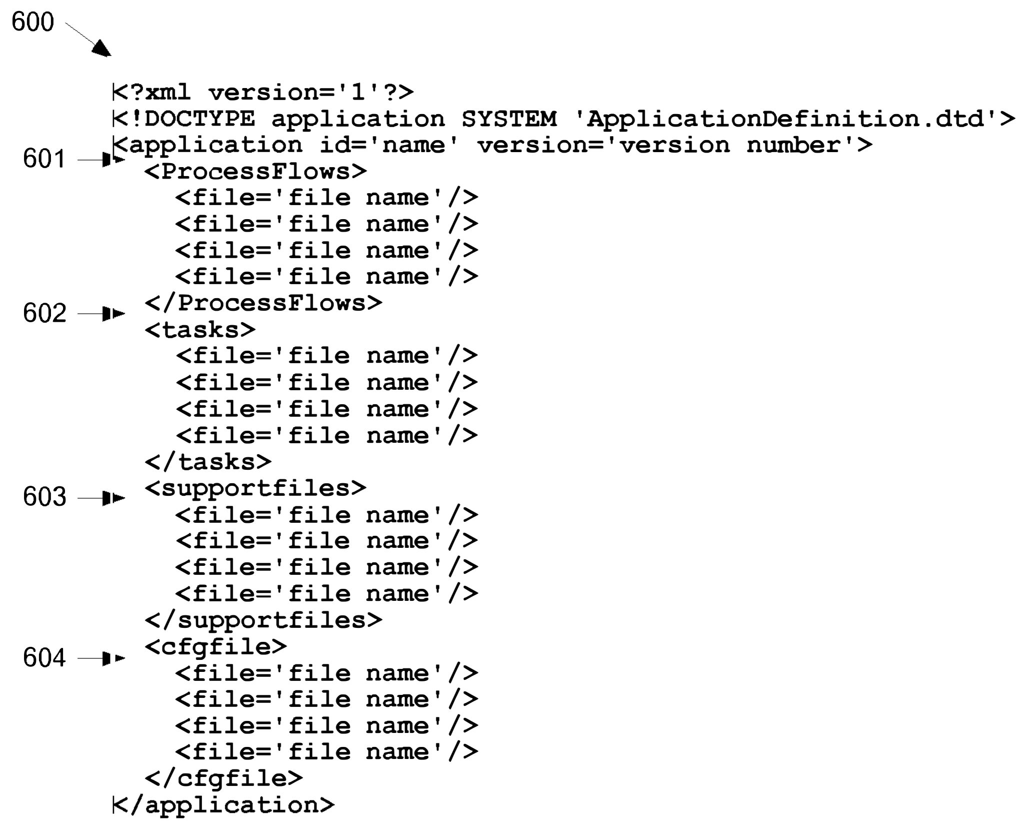
TASK HANDLER PROCESSING

FIGURE 5C



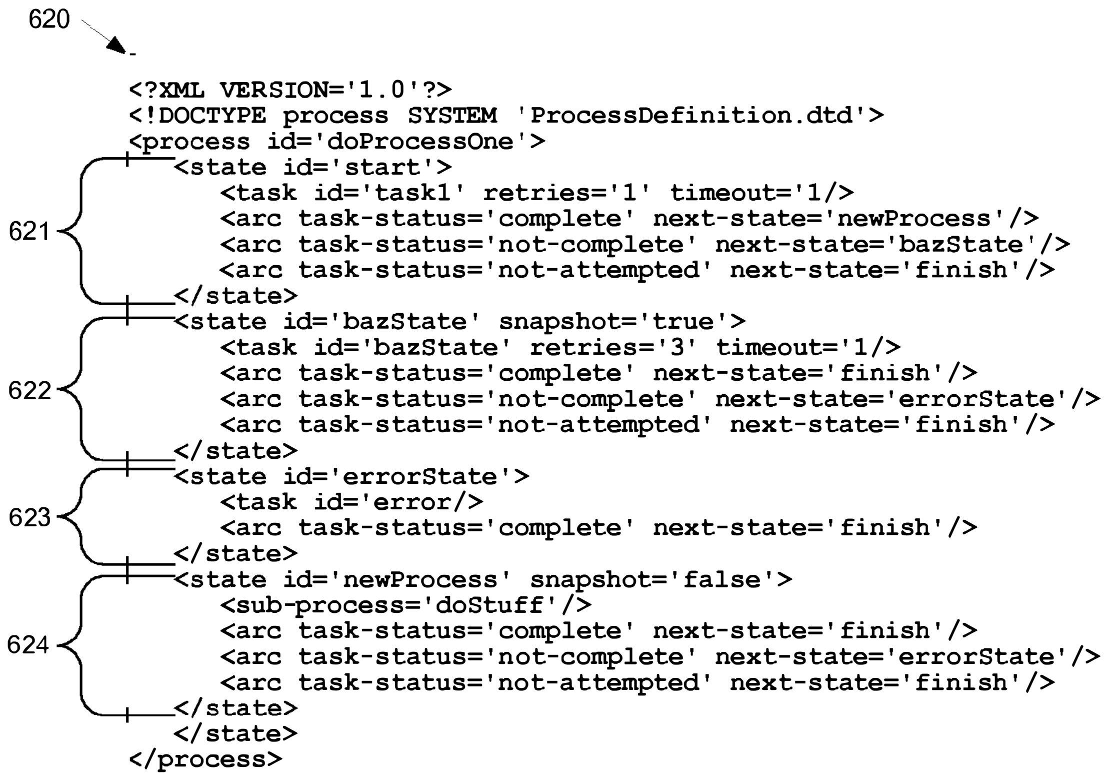
RECOVERY LAYER PROCESSING

FIGURE 5D



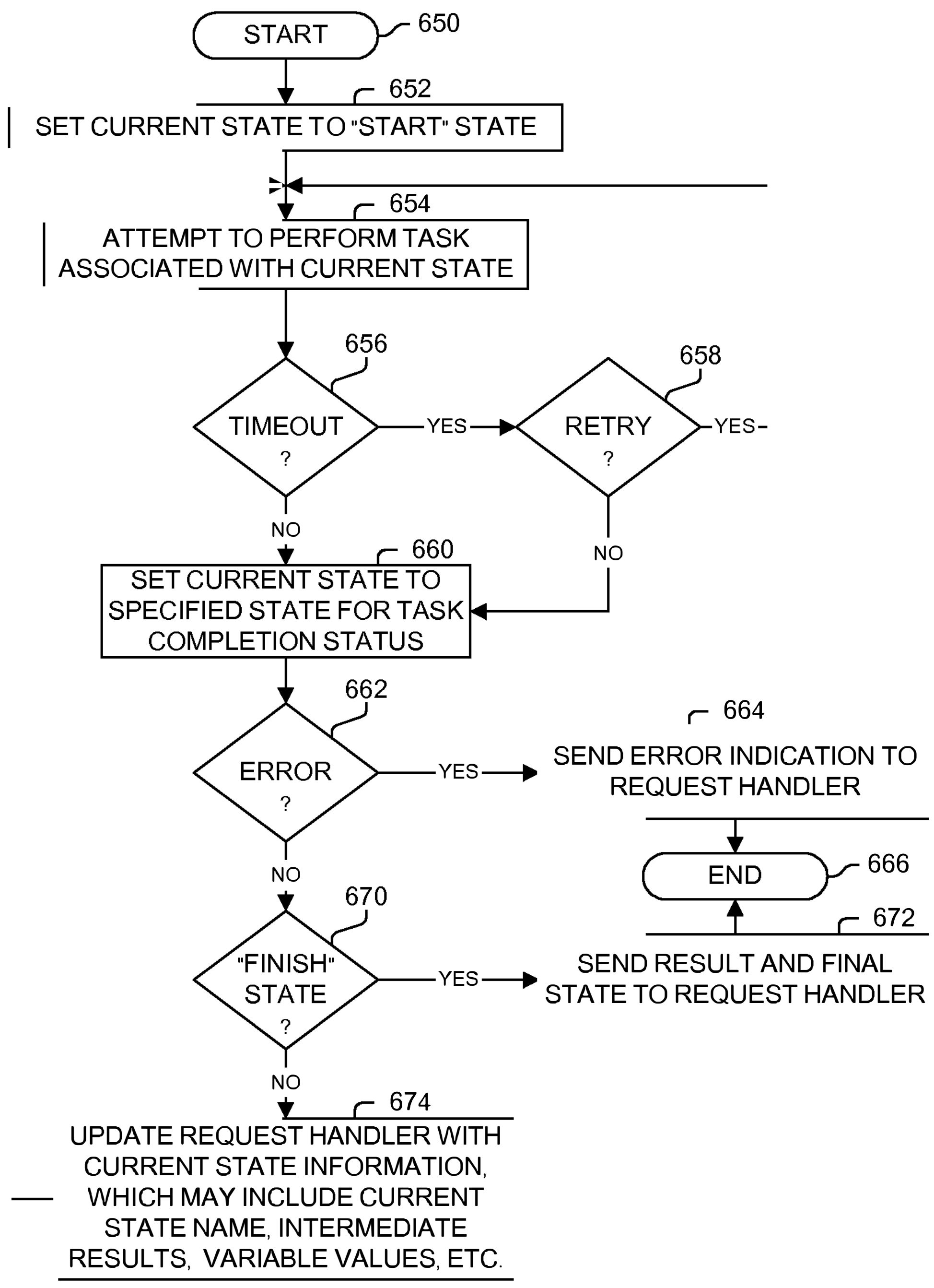
```
<?xml version='1'?>
<!DOCTYPE application SYSTEM 'ApplicationDefinition.dtd'>
<application id='name' version='version number'>
  <ProcessFlows>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </ProcessFlows>
  <tasks>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </tasks>
  <supportfiles>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </supportfiles>
  <cfgfile>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
    <file='file name'/>
  </cfgfile>
</application>
```

FIGURE 6A



```
<?XML VERSION='1.0'?>
<!DOCTYPE process SYSTEM 'ProcessDefinition.dtd'>
<process id='doProcessOne'>
   <state id='start'>
      <task id='task1' retries='1' timeout='1/>
      <arc task-status='complete' next-state='newProcess'/>
      <arc task-status='not-complete' next-state='bazState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   <state id='bazState' snapshot='true'>
      <task id='bazState' retries='3' timeout='1/>
      <arc task-status='complete' next-state='finish'/>
      <arc task-status='not-complete' next-state='errorState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   <state id='errorState'>
      <task id='error/>
      <arc task-status='complete' next-state='finish'/>
   </state>
   <state id='newProcess' snapshot='false'>
      <sub-process='doStuff'/>
      <arc task-status='complete' next-state='finish'/>
      <arc task-status='not-complete' next-state='errorState'/>
      <arc task-status='not-attempted' next-state='finish'/>
   </state>
   </state>
</process>
```

FIGURE 6B


START
650
SET CURRENT STATE TO "START" STATE
652
ATTEMPT TO PERFORM TASK ASSOCIATED WITH CURRENT STATE
654
TIMEOUT ?
656
YES
RETRY ?
658
YES
NO
NO
SET CURRENT STATE TO SPECIFIED STATE FOR TASK COMPLETION STATUS
660
ERROR ?
662
YES
SEND ERROR INDICATION TO REQUEST HANDLER
664
END
666
NO
"FINISH" STATE ?
670
YES
SEND RESULT AND FINAL STATE TO REQUEST HANDLER
672
NO
UPDATE REQUEST HANDLER WITH CURRENT STATE INFORMATION, WHICH MAY INCLUDE CURRENT STATE NAME, INTERMEDIATE RESULTS, VARIABLE VALUES, ETC.
674

EXEMPLARY PROCESS FLOW PROCESSING

FIGURE 6C

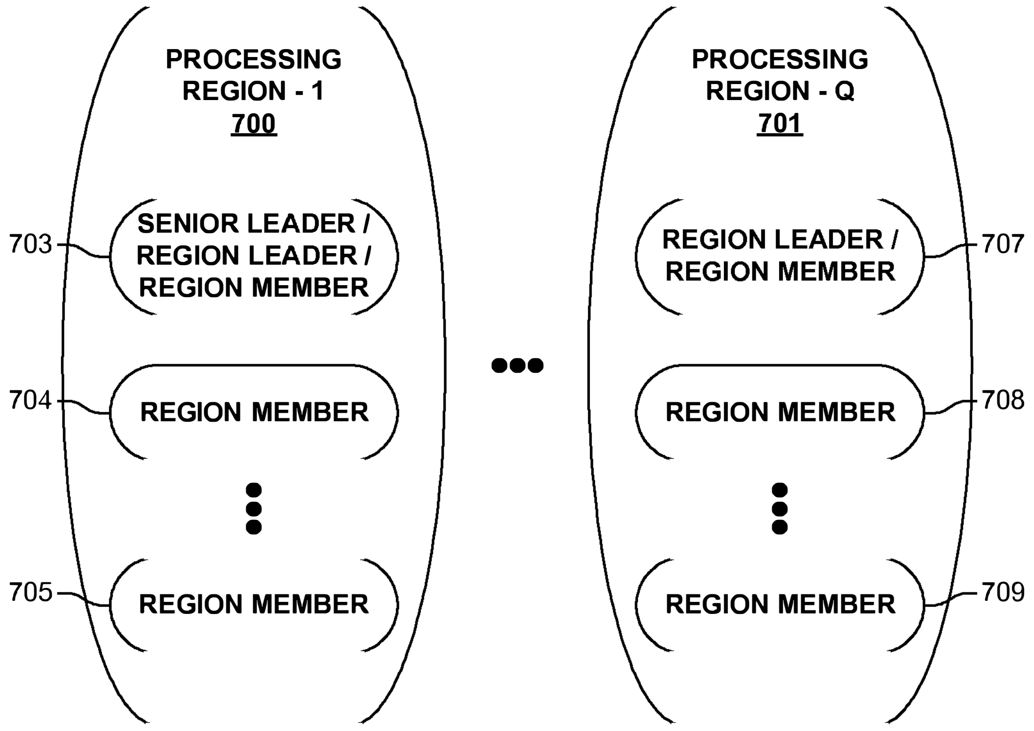
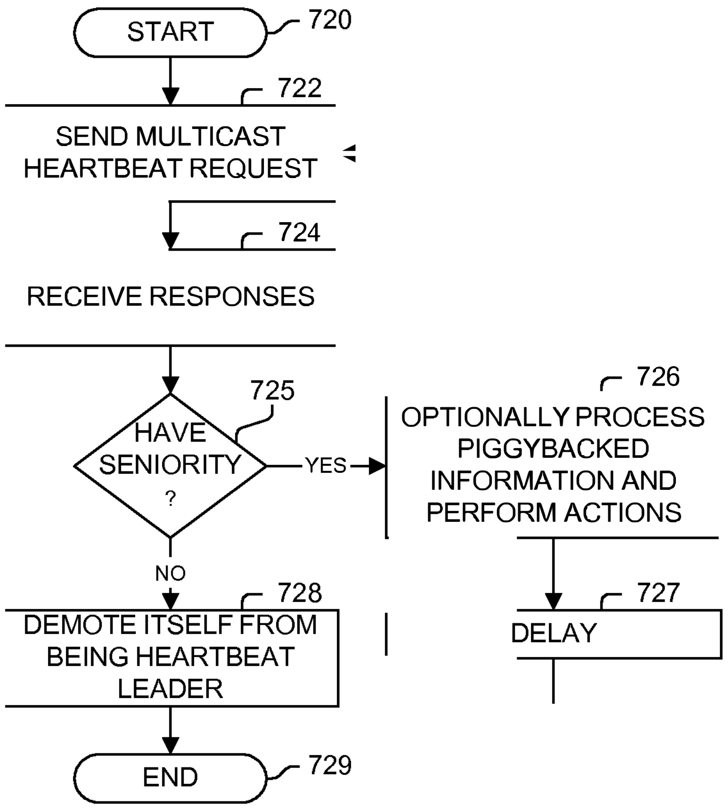
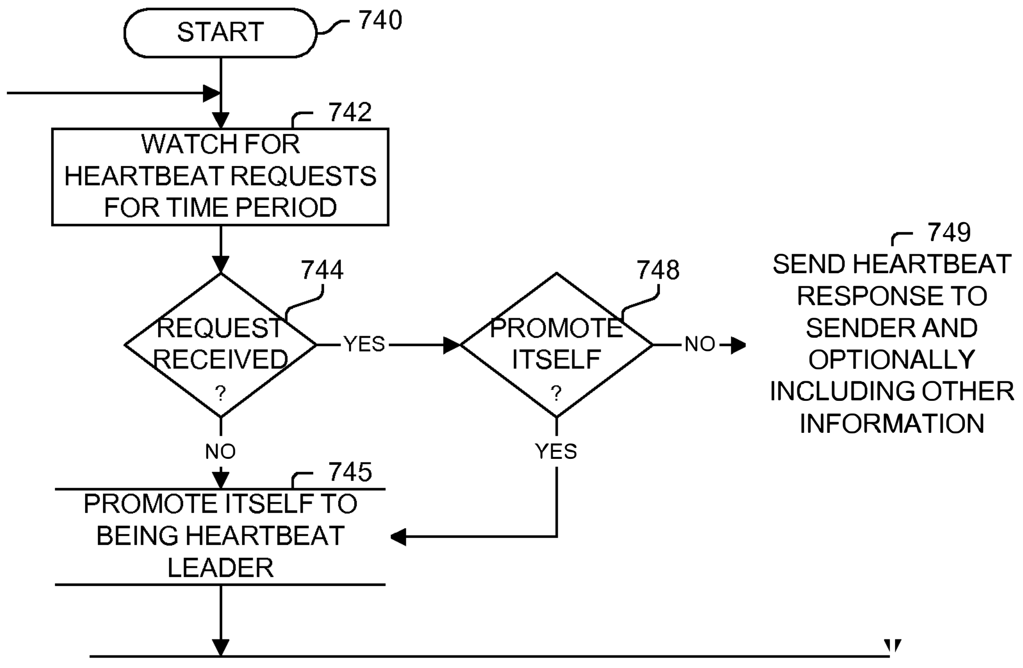

PROCESSING REGION - 1 700
SENIOR LEADER / REGION LEADER / REGION MEMBER
703
REGION MEMBER
704
REGION MEMBER
705
PROCESSING REGION - Q 701
REGION LEADER / REGION MEMBER
707
REGION MEMBER
708
REGION MEMBER
709

FIGURE 7A


START
720
722
SEND MULTICAST HEARTBEAT REQUEST
724
RECEIVE RESPONSES
725
HAVE SENIORITY ?
YES
726
OPTIONALLY PROCESS PIGGYBACKED INFORMATION AND PERFORM ACTIONS
NO
728
DEMOTE ITSELF FROM BEING HEARTBEAT LEADER
727
DELAY
END
729

HEARTBEAT LEADER

FIGURE 7B


START
740
742
WATCH FOR HEARTBEAT REQUESTS FOR TIME PERIOD
744
REQUEST RECEIVED ?
YES
NO
748
PROMOTE ITSELF ?
NO
YES
749
SEND HEARTBEAT RESPONSE TO SENDER AND OPTIONALLY INCLUDING OTHER INFORMATION
745
PROMOTE ITSELF TO BEING HEARTBEAT LEADER

HEARTBEAT MEMBER

FIGURE 7C

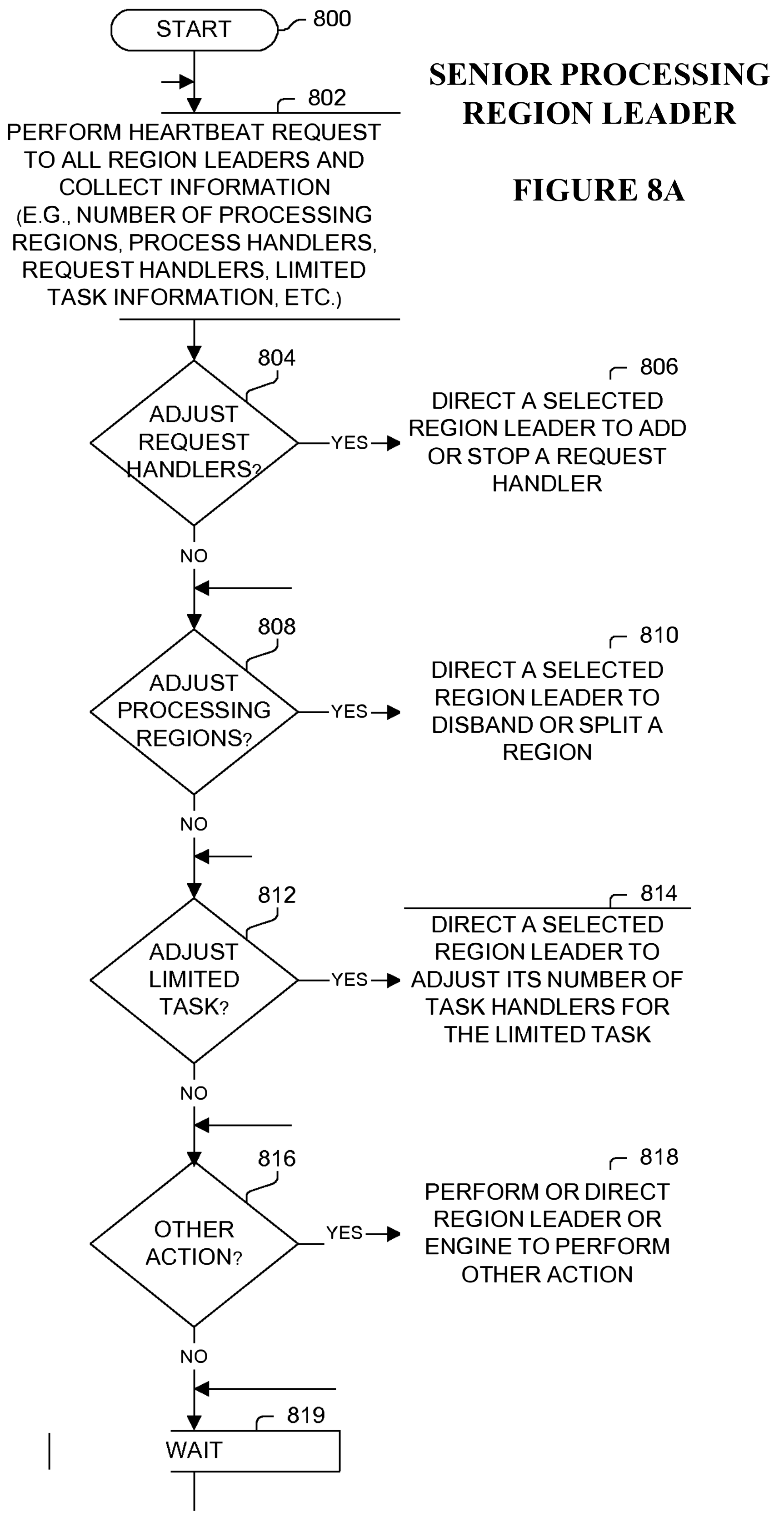
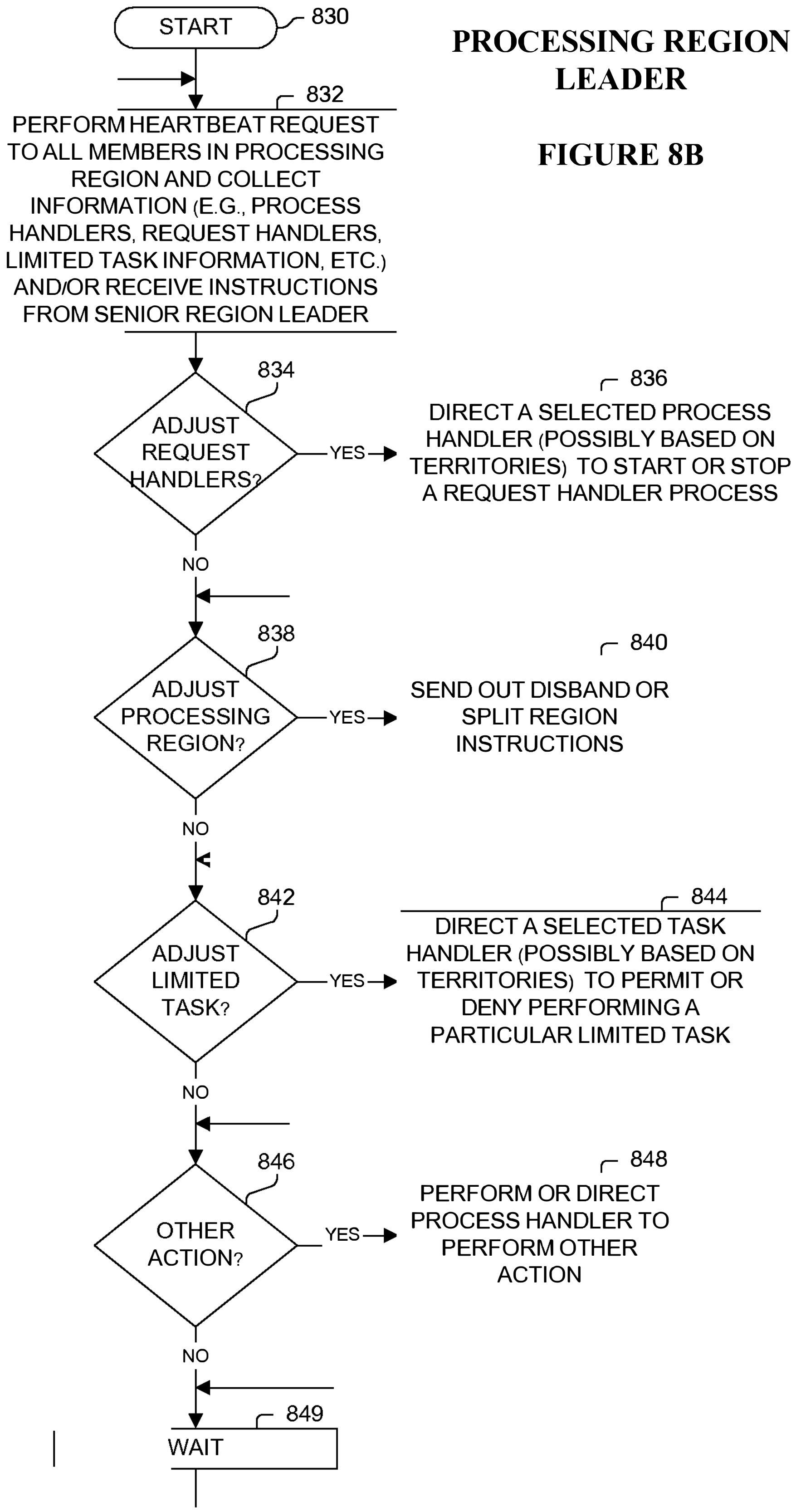
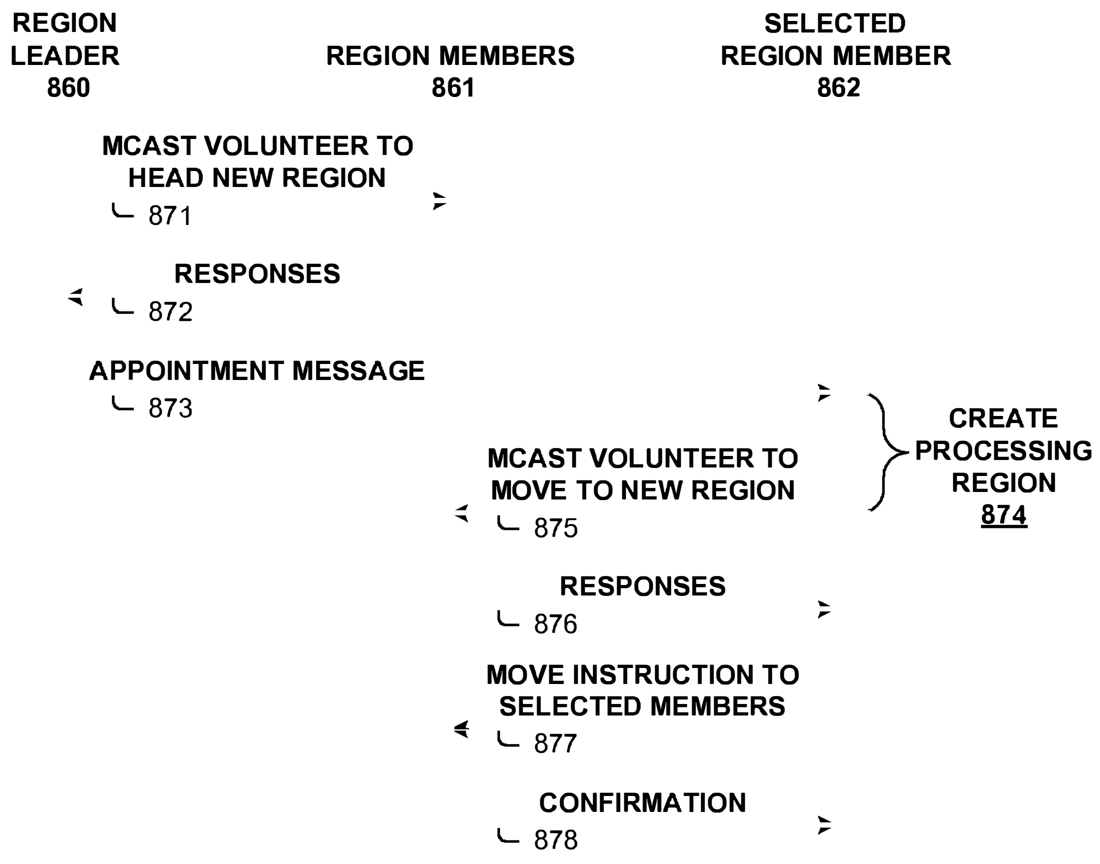

START
800
802
PERFORM HEARTBEAT REQUEST TO ALL REGION LEADERS AND COLLECT INFORMATION (E.G., NUMBER OF PROCESSING REGIONS, PROCESS HANDLERS, REQUEST HANDLERS, LIMITED TASK INFORMATION, ETC.)
SENIOR PROCESSING REGION LEADER
804
ADJUST REQUEST HANDLERS?
YES
806
DIRECT A SELECTED REGION LEADER TO ADD OR STOP A REQUEST HANDLER
NO
808
ADJUST PROCESSING REGIONS?
YES
810
DIRECT A SELECTED REGION LEADER TO DISBAND OR SPLIT A REGION
NO
812
ADJUST LIMITED TASK?
YES
814
DIRECT A SELECTED REGION LEADER TO ADJUST ITS NUMBER OF TASK HANDLERS FOR THE LIMITED TASK
NO
816
OTHER ACTION?
YES
818
PERFORM OR DIRECT REGION LEADER OR ENGINE TO PERFORM OTHER ACTION
NO
819
WAIT

FIGURE 8A

PROCESSING REGION LEADER

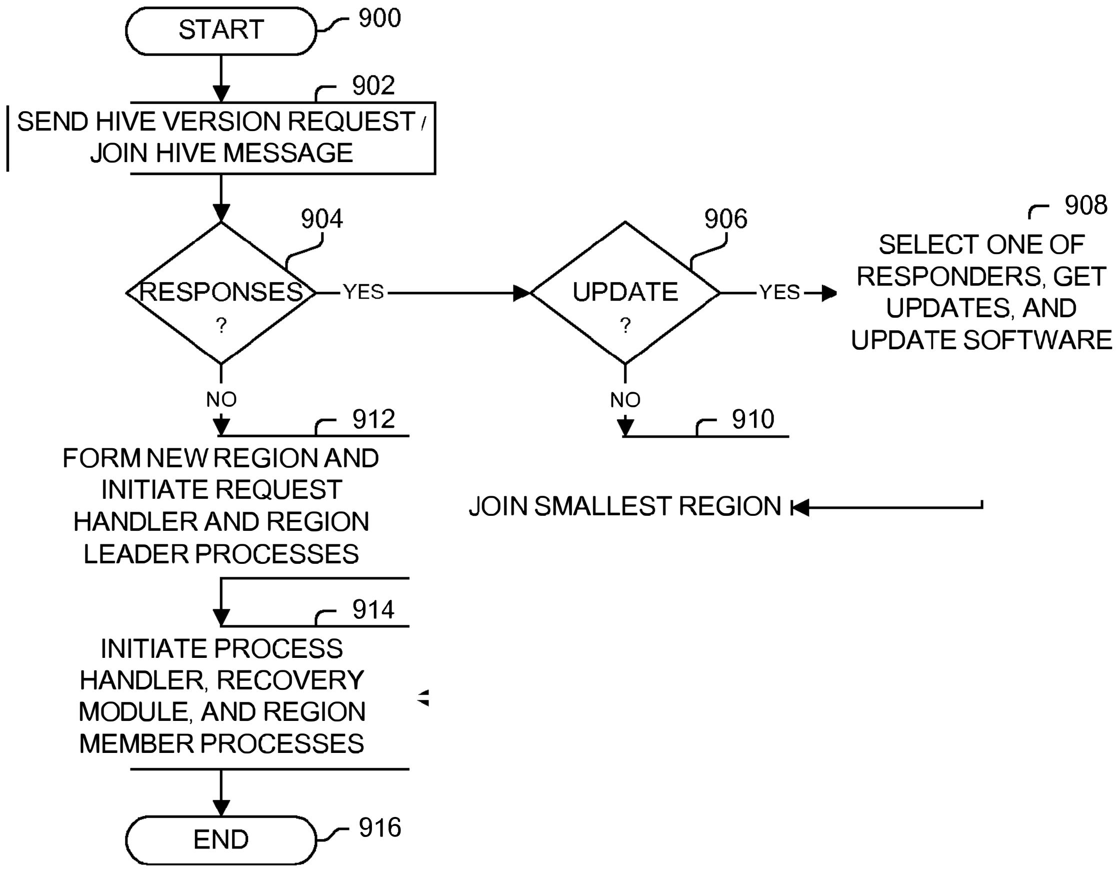
FIGURE 8B


START
830
832
PERFORM HEARTBEAT REQUEST TO ALL MEMBERS IN PROCESSING REGION AND COLLECT INFORMATION (E.G., PROCESS HANDLERS, REQUEST HANDLERS, LIMITED TASK INFORMATION, ETC.) AND/OR RECEIVE INSTRUCTIONS FROM SENIOR REGION LEADER
834
ADJUST REQUEST HANDLERS?
YES
NO
836
DIRECT A SELECTED PROCESS HANDLER (POSSIBLY BASED ON TERRITORIES) TO START OR STOP A REQUEST HANDLER PROCESS
838
ADJUST PROCESSING REGION?
YES
NO
840
SEND OUT DISBAND OR SPLIT REGION INSTRUCTIONS
842
ADJUST LIMITED TASK?
YES
NO
844
DIRECT A SELECTED TASK HANDLER (POSSIBLY BASED ON TERRITORIES) TO PERMIT OR DENY PERFORMING A PARTICULAR LIMITED TASK
846
OTHER ACTION?
YES
NO
848
PERFORM OR DIRECT PROCESS HANDLER TO PERFORM OTHER ACTION
849
WAIT


REGION LEADER 860
REGION MEMBERS 861
SELECTED REGION MEMBER 862
MCAST VOLUNTEER TO HEAD NEW REGION 871
RESPONSES 872
APPOINTMENT MESSAGE 873
CREATE PROCESSING REGION 874
MCAST VOLUNTEER TO MOVE TO NEW REGION 875
RESPONSES 876
MOVE INSTRUCTION TO SELECTED MEMBERS 877
CONFIRMATION 878

REGION SPLITTING

FIGURE 8C


START
900
SEND HIVE VERSION REQUEST / JOIN HIVE MESSAGE
902
RESPONSES ?
904
YES
NO
UPDATE ?
906
YES
NO
SELECT ONE OF RESPONDERS, GET UPDATES, AND UPDATE SOFTWARE
908
FORM NEW REGION AND INITIATE REQUEST HANDLER AND REGION LEADER PROCESSES
912
JOIN SMALLEST REGION
910
INITIATE PROCESS HANDLER, RECOVERY MODULE, AND REGION MEMBER PROCESSES
914
END
916

STARTUP PROCESS

FIGURE 9

SYSTEM AND METHOD FOR PROCESSING INFORMATION VIA NETWORKED COMPUTERS INCLUDING REQUEST HANDLERS, PROCESS HANDLERS, AND TASK HANDLERS

CROSS-REFERENCE AND PRIORITY CLAIM TO RELATED APPLICATIONS

This is a continuation of copending nonprovisional application Ser. No. 13/707,861, filed Dec. 7, 2012, now U.S. Pat. No. 8,682,959, which is a continuation of nonprovisional application Ser. No. 13/491,893, filed Jun. 8, 2012, now U.S. Pat. No. 8,341,209, which is a continuation of nonprovisional application Ser. No. 13/293,527, filed Nov. 10, 2011, now U.S. Pat. No. 8,200,746, which is a divisional of nonprovisional application Ser. No. 12/127,070, filed May 27, 2008, now U.S. Pat. No. 8,060,552, which is a divisional of nonprovisional application Ser. No. 10/236,784, filed Sep. 7, 2002, now U.S. Pat. No. 7,379,959, the entire disclosure of which being hereby incorporated by reference in its entirety.

FIELD OF THE INVENTION

This invention especially relates to processing of information including, but not limited to transactional processing using multiple networked computing systems; and more particularly, the invention relates to processing information using a hive of computing engines, typically including request handlers and process handlers.

INTRODUCTION

Many businesses are demanding faster, less expensive, and more reliable computing platforms. Brokerage houses, credit card processors, telecommunications firms, as well as banks are a few examples of organizations that require tremendous computing power to handle a countless number of small independent transactions. Currently, organizations that require these systems operate and maintain substantial servers. Further, the cost associated with these machines stems not only from the significant initial capital investment, but the continuing expense of a sizeable labor force dedicated to maintenance.

When it comes to mission-critical computing, businesses and other organizations face increasing pressure to do more with less. On one hand, they must manage larger transaction volumes, larger user populations, and larger data sets. They must do all of this in an environment that demands a renewed appreciation for the importance of reliability, fault tolerance, and disaster recovery. On the other hand, they must satisfy these growing requirements in a world of constrained resources. It is no longer an option to just throw large amounts of expensive hardware, and armies of expensive people, at problems. The challenge businesses face is that, when it comes to platforms for mission-critical computing, the world is fragmented. Different platforms are designed to satisfy different sets of requirements. As a result, businesses must choose between, and trade off, equally important factors.

Currently, when it comes to developing, deploying, and executing mission-critical applications, businesses and other organizations can choose between five alternative platforms. These are mainframes, high-availability computers, UNIX-based servers, distributed supercomputers, and PC's. Each of these approaches has strengths and weaknesses, advantages and disadvantages.

The first, and oldest, solution to the problem of mission-critical computing was the mainframe. Mainframes dominated the early days of computing because they delivered both availability and predictability. Mainframes deliver availability because they are located in extremely controlled physical environments and are supported by large cadres of dedicated, highly-trained people. This helps to ensure they do not fall victim to certain types of problems. However, because they are typically single-box machines, mainframes remain vulnerable to single-point failures. Mainframes deliver predictability because it is possible to monitor the execution and completion of processes and transactions and restart any that fail. However, the limitation of mainframes is that all monitoring code must be understood, written, and/or maintained by the application developer. The problem mainframes run into is that such systems fall short when it comes to three factors of high importance to businesses. First, mainframes tend not to offer high degrees of scalability. The only way to significantly increase the capability of such a system is to buy a new one. Second, because of their demanding nature, mainframes rely on armies of highly-trained support personnel and custom hardware. As a result, mainframes typically are neither affordable nor maintainable.

Developed to address the limitations and vulnerabilities of mainframes, high-availability computers are able to offer levels of availability and predictability that are equivalent to, and often superior to, mainframes. High-availability computers deliver availability because they use hardware or software-based approaches to ensure high levels of survivability. However, this availability is only relative because such systems are typically made up of a limited number of components. High-availability computers also deliver predictability because they offer transaction processing and monitoring capabilities. However, as with mainframes, that monitoring code must be understood, written, and/or maintained by the application developer. The problem with high-availability computers is that have many of the same shortcomings as mainframes. That means that they fall short when it comes to delivering scalability, affordability, and maintainability. First, they are largely designed to function as single-box systems and thus offer only limited levels of scalability. Second, because they are built using custom components, high-availability computers tend not to be either affordable or maintainable.

UNIX-based servers are scalable, available, and predictable but are expensive both to acquire and to maintain. Distributed supercomputers, while delivering significant degrees of scalability and affordability, fall short when it comes to availability. PC's are both affordable and maintainable, but do not meet the needs of businesses and other organizations when it comes to scalability, availability, and predictability. The 1990s saw the rise of the UNIX-based server as an alternative to mainframes and high-availability computers. These systems have grown in popularity because, in addition to delivering availability and predictability, they also deliver significant levels of scalability. UNIX-based servers deliver degrees of scalability because it is possible to add new machines to a cluster and receive increases in processing power. They also deliver availability because they are typically implemented as clusters and thus can survive the failure of any individual node. Finally, UNIX-based servers deliver some degree of predictability. However, developing this functionality can require significant amounts of custom development work.

One problem that UNIX-based servers run into, and the thing that has limited their adoption, is that this functionality comes at a steep price. Because they must be developed and maintained by people with highly specialized skills, they fall short when it comes to affordability and maintainability. For one thing, while it is theoretically possible to build a UNIX-based server using inexpensive machines, most are still implemented using small numbers of very expensive boxes. This makes upgrading a UNIX-based server an expensive and time-consuming process that must be performed by highly-skilled (and scarce) experts. Another limitation of UNIX-based servers is that developing applications for them typically requires a significant amount of effort. This requires application developers to be experts in both the UNIX environment and the domain at hand. Needless to say, such people can be hard to find and are typically quite expensive. Finally, setting up, expanding, and maintaining a UNIX-based server requires a significant amount of effort on the part of a person intimately familiar with the workings of the operating system. This reflects the fact that most were developed in the world of academia (where graduate students are plentiful). However, this can create significant issues for organizations that do not have such plentiful supplies of cheap, highly-skilled labor.

A recent development in the world of mission-critical computing is the distributed supercomputer (also known as a Network of Workstations or "NOW"). A distributed supercomputer is a computer that works by breaking large problems up into a set of smaller ones that can be spread across many small computers, solved independently, and then brought back together. Distributed supercomputers were created by academic and research institutions to harness the power of idle PC and other computing resources. This model was then adapted to the business world, with the goal being to make use of underused desktop computing resources. The most famous distributed supercomputing application was created by the Seti@Home project. Distributed supercomputers have grown in popularity because they offer both scalability and affordability. Distributed supercomputers deliver some degree of scalability because adding an additional resource to the pool usually yields a linear increase in processing power. However that scalability is limited by the fact that communication with each node takes place over the common organizational network and can become bogged down. Distributed supercomputers are also relatively more affordable than other alternatives because they take advantage of existing processing resources, be they servers or desktop PC's.

One problem distributed supercomputers run into is that they fall short when it comes to availability, predictability, and maintainability. Distributed supercomputers have problems delivering availability and predictability because they are typically designed to take advantage of non-dedicated resources. The problem is that it is impossible to deliver availability and predictability when someone else has primary control of the resource and your application is simply completing its work when it gets the chance. This makes distributed supercomputers appropriate for some forms of off-peak processing but not for time-sensitive or mission-critical computing. Finally, setting up, expanding, and maintaining a distributed supercomputer also requires a significant amount of effort because they tend to offer more of a set of concepts than a set of tools. As a result, they require significant amounts of custom coding. Again, this reflects the fact that most were developed in the world of academia where highly trained labor is both cheap and plentiful.

PC's are another option for creating mission-critical applications. PC's have two clear advantages relative to other solutions. First, PC's are highly affordable. The relentless progress of Moore's law means that increasingly powerful PC's can be acquired for lower and lower prices. The other advantage of PC's is that prices have fallen to such a degree

that many people have begun to regard PC's as disposable. Given how fast the technology is progressing, in many cases it makes more sense to replace a PC than to repair it. Of course, the problem with PC's is that they do not satisfy the needs of businesses and other organizations when it comes to scalability, availability, and predictability. First, because PC's were designed to operate as stand-alone machines, they are not inherently scalable. Instead, the only way to allow them to scale is to link them together into clusters. That can be a very time-consuming process. Second, PC's, because they were designed for use by individuals, were not designed to deliver high levels of availability. As a result, the only way to make a single PC highly available is through the use of expensive, custom components. Finally, PC's were not designed to handle transaction processing and thus do not have any provisions for delivering predictability. The only way to deliver this functionality is to implement it using the operating system or an application server. The result is that few organizations even consider using PC's for mission-critical computing.

In a dynamic environment, it is important to be able to find available services. Service Location Protocol, RFC 2165, June 1997, provides one such mechanism. The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet no longer need so much static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration. The basic operation in Service Location is that a client attempts to discover the location of a Service. In smaller installations, each service will be configured to respond individually to each client. In larger installations, services will register their services with one or more Directory Agents, and clients will contact the Directory Agent to fulfill requests for Service Location information. Clients may discover the whereabouts of a Directory Agent by preconfiguration, DHCP, or by issuing queries to the Directory Agent Discovery multicast address.

The following describes the operations a User Agent would employ to find services on the site's network. The User Agent needs no configuration to begin network interaction. The User Agent can acquire information to construct predicates which describe the services that match the user's needs. The User Agent may build on the information received in earlier network requests to find the Service Agents advertising service information.

A User Agent will operate two ways. First, if the User Agent has already obtained the location of a Directory Agent, the User Agent will unicast a request to it in order to resolve a particular request. The Directory Agent will unicast a reply to the User Agent. The User Agent will retry a request to a Directory Agent until it gets a reply, so if the Directory Agent cannot service the request (say it has no information) it must return an response with zero values, possibly with an error code set.

Second, if the User Agent does not have knowledge of a Directory Agent or if there are no Directory Agents available on the site network, a second mode of discovery may be used. The User Agent multicasts a request to the service-specific multicast address, to which the service it wishes to locate will respond. All the Service Agents which are listening to this multicast address will respond, provided they can satisfy the User Agent's request. A similar mechanism is used for Directory Agent discovery. Service Agents which have no information for the User Agent MUST NOT respond.

While the multicast/convergence model may be important for discovering services (such as Directory Agents) it is the exception rather than the rule. Once a User Agent knows of the location of a Directory Agent, it will use a unicast request/response transaction. The Service Agent SHOULD listen for multicast requests on the service-specific multicast address, and MUST register with an available Directory Agent. This Directory Agent will resolve requests from User Agents which are unicasted using TCP or UDP. This means that a Directory Agent must first be discovered, using DHCP, the DA Discovery Multicast address, the multicast mechanism described above, or manual configuration. If the service is to become unavailable, it should be deregistered with the Directory Agent. The Directory Agent responds with an acknowledgment to either a registration or deregistration. Service Registrations include a lifetime, and will eventually expire. Service Registrations need to be refreshed by the Service Agent before their Lifetime runs out. If need be, Service Agents can advertise signed URLs to prove that they are authorized to provide the service.

New mechanisms for computing are desired, especially those which may provide a reliable computing framework and platform, including, but not limited to those which might produce improved levels of performance and reliability at a much lower cost than that of other solutions.

SUMMARY

A hive of computing engines, typically including request handlers and process handlers, is used to process information. One embodiment includes a request region including multiple request handlers and multiple processing regions, each typically including multiple process handlers. Each request handler is configured to respond to a client service request of a processing job, and if identified to handle the processing job: to query one or more of the processing regions to identify and assign a particular process handler to service the processing job, and to receive a processing result from the particular process handler. Each of the process handlers is configured to respond to such a query, and if identified as the particular process handler: to service the processing job, to process the processing job, to update said identified request handler with state information pertaining to partial processing of said processing job, and to communicate the processing result to the identified request handler. One embodiment includes multiple task handlers, wherein a process handler assigns a task identified with the processing job to one of task handlers, which performs the task and returns the result. In one embodiment, the selection of a task handler to perform a particular task is determined based on a volunteer pattern initiated by the process handler.

Another exemplary embodiment comprises a system for processing information, the system comprising a plurality of networked computers for processing a plurality of processing jobs in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow, wherein the request handler is configured to (1) receive a plurality of service requests for the processing jobs, (2) store state information for the processing jobs, and (3) communicate data relating to the processing jobs to a plurality of the process handlers, wherein the process handlers to which the data relating to the processing jobs were communicated are configured to (1) analyze the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identify a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determine that the processing job corresponding to the process flow with no remaining processing tasks has been completed, wherein the task handlers are configured to perform the identified processing tasks to generate a plurality of task results, and wherein the request handler is further configured to store updated state information for the processing jobs, the updated stored state information being based on the task results.

Still another exemplary embodiment comprises a method for processing information, the method comprising: (a) receiving a service request for a processing job, the processing job having an associated process flow, the process flow including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the process flow, and (b) executing the processing job in a distributed manner by a plurality of networked computers and in accordance with the received service request, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, wherein the executing step comprises: (i) the request handler storing state information for the processing job, (ii) the request handler communicating data for the processing job to a process handler, (iii) the process handler to which the data for the processing job was communicated (1) analyzing the state information for the processing job to determine whether any processing task in the process flow remains to be performed based on the logic for the process flow, (2) in response to the state information analysis indicating that a processing task remains for the process flow, identifying a processing task to be performed, and (3) in response to the state information analysis indicating that no processing task remains for the process flow, determining that the processing job has been completed, (iv) the task handlers performing the identified processing tasks to generate a plurality of task results, and (v) updating the stored state information based on the task results.

Yet another exemplary embodiment comprises a method for processing information, the method comprising: (a) receiving a plurality of service requests for a plurality of processing jobs, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow, and (b) executing the processing jobs in a distributed manner by a plurality of networked computers and in accordance with the received service requests, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, wherein the executing step comprises: (i) the request handler storing state information for the processing jobs, (ii) the request handler communicating data relating to the processing jobs to a plurality of the process handlers, (iii) the process handlers to which the data relating to the processing jobs were communicated (1) analyzing the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identifying a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determining that the processing job corresponding to the process flow with no remaining processing tasks has been completed, (iv) the task handlers performing the identified processing tasks to generate a plurality of task results, and (v) updating the stored state information based on the task results.

Still another exemplary embodiment comprises a system for processing information, the system comprising a plurality of networked computers for processing a plurality of processing jobs in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow, wherein the request handler is configured to (1) receive a plurality of service requests for the processing jobs, and (2) store state information for the processing jobs, wherein the process handlers are configured to volunteer for servicing the processing jobs based on their availabilities, wherein the request handler is further configured to communicate data relating to the processing jobs to a plurality of the process handlers that volunteered, wherein the process handlers to which the data relating to the processing jobs were communicated are configured to (1) analyze the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identify a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determine that the processing job corresponding to the process flow with no remaining processing tasks has been completed, wherein the task handlers are configured to volunteer for performing tasks based on their availabilities, wherein a plurality of the task handlers that volunteered are configured to perform the identified processing tasks to generate a plurality of task results, and wherein the request handler is further configured to store updated state information for the processing jobs, the updated stored state information being based on the task results.

BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the invention with particularity. The invention, together with its advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

FIG. **1**A illustrates an architecture of hives used in one embodiment;

FIG. **1**B illustrates a computing platform used for a hive engine for implementing request handlers, process handlers, and/or other processes of a hive of one embodiment, or also used for simulating the operation of a hive in one embodiment;

FIG. **2**A illustrates a hierarchy of a hive, request regions, territories, and processing regions as used in one embodiment;

FIG. **2**B illustrates an interaction of a client, request handlers, and process handlers of one embodiment;

FIG. **2**C illustrates multicast addresses used in one embodiment;

FIG. **2**D illustrates the flow of messages between components of one embodiment;

FIG. **2**E illustrates an interaction of a client, request handlers, process handlers and possibly tasks of one embodiment;

FIG. **3** is a flow diagram of a client process used in one embodiment;

FIGS. **4**A-C are flow diagrams of request hander processes used in one embodiment;

FIG. **5**A-B are flow diagrams of process hander processes used in one embodiment;

FIG. **5**C is a flow diagram of a task handler process used in one embodiment;

FIG. **5**D is a flow diagram of a recovery layer process used in one embodiment;

FIG. **6**A illustrates a definition of an application used in one embodiment;

FIG. **6**B illustrates a definition of an process flow used in one embodiment;

FIG. **6**C illustrates a process used in one embodiment for executing a process flow;

FIG. **7**A illustrates a hierarchy of a senior region leaders, region leaders, and region members among multiple processing regions as used in one embodiment;

FIGS. **7**B-**7**C are flow diagrams of processes used in one embodiment to establish and maintain a hierarchical relationship among distributed processes;

FIG. **8**A is a flow diagram of a senior processing region leader process used in one embodiment;

FIG. **8**B is a flow diagram of a processing region leader process used in one embodiment;

FIG. **8**C illustrates the splitting of a region as performed in one embodiment; and

FIG. **9** illustrates a process used in one embodiment for initializing a hive engine.

DETAILED DESCRIPTION

A hive of computing engines, typically including request handlers and process handlers, is used to process information. Each of the claims individually recites an aspect of the invention in its entirety. Moreover, some embodiments described may include, but are not limited to, inter alia, systems, networks, integrated circuit chips, embedded processors, ASICs, methods, apparatus, and computer-readable medium containing instructions. The embodiments described hereinafter embody various aspects and configurations within the scope and spirit of the invention, with the figures illustrating exemplary and non-limiting configurations.

The term "system" is used generically herein to describe any number of components, elements, sub-systems, devices, packet switch elements, packet switches, routers, networks, computer and/or communication devices or mechanisms, or combinations of components thereof. The term "computer" is used generically herein to describe any number of computers, including, but not limited to personal computers, embedded processing elements and systems, control logic, ASICs, chips, workstations, mainframes, etc. The term "processing element" is used generically herein to describe any type of processing mechanism or device, such as a processor, ASIC, field programmable gate array, computer, etc. The term "device" is used generically herein to describe any type of mechanism, including a computer or system or component thereof. The terms "task" and "process" are used generically herein to describe any type of running program, including, but not limited to a computer process, task, thread, executing application, operating system, user process, device driver, native code, machine or other language, etc., and can be interactive and/or non-interactive, executing locally and/or remotely, executing in foreground and/or background, executing in the user and/or operating system address spaces, a routine of a library and/or standalone application, and is not limited to any particular memory partitioning technique. The steps, connections, and processing of signals and information illustrated in the figures, including, but not limited to any block and flow diagrams and message sequence charts, may be performed in the same or in a different serial or parallel ordering and/or by different components and/or processes, threads, etc., and/or over different connections and be combined with other functions in other embodiments in keeping within the scope and spirit of the invention. Furthermore, the term "identify" is used generically describe any manner or mechanism for directly or indirectly ascertaining something, which may include, but is not limited to receiving, retrieving from memory, determining, calculating, generating, etc.

Moreover, the terms "network" and "communications mechanism" are used generically herein to describe one or more networks, communications mediums or communications systems, including, but not limited to the Internet, private or public telephone, cellular, wireless, satellite, cable, local area, metropolitan area and/or wide area networks, a cable, electrical connection, bus, etc., and internal communications mechanisms such as message passing, interprocess communications, shared memory, etc. The term "message" is used generically herein to describe a piece of information which may or may not be, but is typically communicated via one or more communication mechanisms of any type, such as, but not limited to a packet.

As used herein, the term "packet" refers to packets of all types or any other units of information or data, including, but not limited to, fixed length cells and variable length packets, each of which may or may not be divisible into smaller packets or cells. The term "packet" as used herein also refers to both the packet itself or a packet indication, such as, but not limited to all or part of a packet or packet header, a data structure value, pointer or index, or any other part or identification of a packet. Moreover, these packets may contain one or more types of information, including, but not limited to, voice, data, video, and audio information. The term "item" is used herein to refer to a packet or any other unit or piece of information or data. The phrases "processing a packet" and "packet processing" typically refer to performing some steps or actions based on the packet, and which may or may not include modifying and/or forwarding the packet.

The term "storage mechanism" includes any type of memory, storage device or other mechanism for maintaining instructions or data in any format. "Computer-readable medium" is an extensible term including any memory, storage device, storage mechanism, and any other storage and signaling mechanisms including interfaces and devices such

as network interface cards and buffers therein, as well as any communications devices and signals received and transmitted, and other current and evolving technologies that a computerized system can interpret, receive, and/or transmit. The term "memory" includes any random access memory (RAM), read only memory (ROM), flash memory, integrated circuits, and/or other memory components or elements. The term "storage device" includes any solid state storage media, disk drives, diskettes, networked services, tape drives, and other storage devices. Memories and storage devices may store computer-executable instructions to be executed by a processing element and/or control logic, and data which is manipulated by a processing element and/or control logic. The term "data structure" is an extensible term referring to any data element, variable, data structure, data base, and/or one or more or an organizational schemes that can be applied to data to facilitate interpreting the data or performing operations on it, such as, but not limited to memory locations or devices, sets, queues, trees, heaps, lists, linked lists, arrays, tables, pointers, etc. A data structure is typically maintained in a storage mechanism. The terms "pointer" and "link" are used generically herein to identify some mechanism for referencing or identifying another element, component, or other entity, and these may include, but are not limited to a reference to a memory or other storage mechanism or location therein, an index in a data structure, a value, etc.

The term "one embodiment" is used herein to reference a particular embodiment, wherein each reference to "one embodiment" may refer to a different embodiment, and the use of the term repeatedly herein in describing associated features, elements and/or limitations does not establish a cumulative set of associated features, elements and/or limitations that each and every embodiment must include, although an embodiment typically may include all these features, elements and/or limitations. In addition, the phrase "means for xxx" typically includes computer-readable medium containing computer-executable instructions for performing xxx.

In addition, the terms "first," "second," etc. are typically used herein to denote different units (e.g., a first element, a second element). The use of these terms herein does not necessarily connote an ordering such as one unit or event occurring or coming before the another, but rather provides a mechanism to distinguish between particular units. Additionally, the use of a singular tense of a noun is non-limiting, with its use typically including one or more of the particular item rather than just one (e.g., the use of the word "memory" typically refers to one or more memories without having to specify "memory or memories," or "one or more memories" or "at least one memory", etc.) Moreover, the phrases "based on x" and "in response to x" are used to indicate a minimum set of items x from which something is derived or caused, wherein "x" is extensible and does not necessarily describe a complete list of items on which the operation is performed, etc. Additionally, the phrase "coupled to" is used to indicate some level of direct or indirect connection between two elements or devices, with the coupling device or devices modify or not modifying the coupled signal or communicated information. The term "subset" is used to indicate a group of all or less than all of the elements of a set. Moreover, the term "or" is used herein to identify a selection of one or more, including all, of the conjunctive items.

Numerous means for processing information using a hive of computing/hive engines are disclosed. One implementation includes a request region including multiple request handlers and multiple processing regions, each typically including multiple process handlers. Each request handler is configured to respond to a client service request of a processing job, and if identified to handle the processing job: to query one or more of the processing regions to identify and assign a particular process handler to service the processing job, and to receive a processing result from the particular process handler. As typically used herein, a result corresponds to the outcome of a successfully or unsuccessfully completed job, task or other operation or an error condition, and typically includes one or more indications of a final value or outcome and/or state information (e.g., indications of to the processing performed or not performed, partial or final results, error descriptors, etc.) Each of the process handlers is configured to respond to such a query, and if identified as the particular process handler: to service the processing job, to process the processing job, to update said identified request handler with state information pertaining to partial processing of said processing job, and to communicate the processing result to the identified request handler.

In one embodiment, a volunteer pattern allows a software application (e.g., client process, request handler, process handler, task handler, tasks, or another hive engine process, etc.) to automatically detect a group of software applications on the same network, and to select and communicate with the most appropriate application without any prior knowledge to the location and capabilities of the chosen software application. In one embodiment, messages are sent among processes typically using multicast UDP, unicast UDP, and standard TCP connections.

In one embodiment, the volunteer pattern includes the following steps. First, hive engines that wish to volunteer its capabilities begin by listening for volunteer requests on a known multicast address. Next, a client looking for a request handler to handle its request transmits its needs by issuing a volunteer or service request packet. The service request packet is a small text buffer which includes the type of service it is requesting and any potential parameters of that request. The service request packet also includes the return IP address of the client for hive engines to use to communicate their volunteer responses. The volunteer packet is communicated via multicast to the known multicast group corresponding to the request region. Request handlers of multiple hive engines on the client's network will detect this request. Third, hive engines that receive the service request packet examine its contents. If the hive engine is capable of servicing this request, it responds by sending a response (e.g., a UDP packet) to the client which made the request. The UDP packet typically contains the TCP address of the hive engine's communication port. Unicast UDP packets are used so that only the client that initiated the service request will receive the volunteer responses from the request handlers. Fourth, the client receives unicast UDP packets from the hive engines, selects one, and connects to the hive engine via TCP socket. The client and hive engine will typically use this socket for all subsequent communications during the processing of this application.

In one embodiment, regionalization is used to allow participating hive engines on the same network to detect each other and organize into logical groups of processing regions without any prior configuration to minimize bandwidth usage and CPU consumption in the entire system. Regionalization provides an automated mechanism that allows these processing regions grow and split as needed, which may provide for an unlimited growth of a hive. Thus, volunteer requests (e.g., processing requests, task requests, etc.) can be within a processing region without affecting all hive engines sending these requests or other communications using a multicast address assigned to a specific processing region. This places a bound on the number of responses to be generated (e.g., by the number of hive engines in a processing region.)

Typically, hive engines participate in an automated self-organization mechanisms, which allows participating hive engines on the same local or wide area network to detect each other and organize into logical groups without any prior configuration. However, an embodiment may use any mechanism for defining a regionalization, or even one embodiment does not use regionalization. For example, in one embodiment, a hive engine is pre-configured with parameters to define which region or regions in which to participate; while in one embodiment, users or a centralized control system is used to specify to one or more hive engines which region or regions in which to participate.

A hive typically has multiple processing regions and a single request region; although, one embodiment includes multiple request regions and one or more processing regions. One way to view a processing region is that it is a set of processes on one or more hive engines for executing processing jobs. In one embodiment, a processing region has a leader that keeps track of the number of hive engines in the region. If the number of hive engines in the region reaches the user defined maximum, the region leader instructs the hive engines in the region to divide into two separate smaller regions. If the number of hive engines in the regions reaches the user defined minimum, the region leader instructs the hive engines in the region to join other regions in the hive.

In one embodiment, the processing regions are self-healing in that if the region leader shuts down for any reason all the region members detect the lack of a region leader. A region member promotes itself to region leader. If a processing region has multiple region leaders, the youngest region leaders demotes themselves back to region members, leaving one region leader.

A request region typically hides that the hive consists of multiple regions and directs the processing load across all the regions. From one perspective, spreading the request region across multiple hive engines provides an increased level of fault tolerance, as these services detect the loss of a connection and rebuild or shutdown as necessary. The hive recovers most failure cases, however, when a request is in an indeterminate state, the request is typically terminated to prevent multiple executions.

In one embodiment, a single senior region leader forms the request region. The senior region leader discovers the region leaders via the volunteer pattern. The senior region leader discovers the size of the request region by asking the region leaders for the number of hive engines in their region that are also members of the request region. If the request region has too many or too few members, the senior region leader directs the region leaders to re-allocate the hive engines to or from the request region. The request region is typically self-healing in that if the senior region leader shuts down for any reason all the region leaders detect the lack of a senior region leader. A region leader promotes itself to senior region leader. If the new senior region leader is not the most senior region leader, the senior region leader demotes itself and the most senior region leader promotes itself to senior region leader. If more than one senior region leader exists, the senior region leaders that are less senior or junior to another senior region leader demotes itself.

In one embodiment, a client processing job is specified in terms of a process flow, typically specifying a set of tasks as well state variables typically before and after each task for storing state information. The hive process flow contains the information on the sequence of sub-routines to be called, timeout and retry information if the sub-routines fail, and which sub-routine to call next based on the sub-routine's result. Once specified, it is up to the hive software to execute the sub-routines in the process flow. A process flow may described in any manner or format. For example, in one embodiment, a process flow is described in a XML process definition file. The process flow definition file defines the process flow name, the task to be performed, the task's recovery procedure including the timeout limit and retry limit, and the transition from one state to the next state based on the previous task's result.

In order to maintain high-availability and fault tolerance, a client processing job is typically performed using a self-organized, non-administered, network of services across several hive engines that work together to guarantee execution of a request even in the event that any of the individual services or hive engines fail. For example, in one embodiment, a processing job is received by a request handler from a client using the volunteer pattern. The request engine selects a process handler based on pattern. The process handler proceeds to perform the processing job, and at intermediate steps within the process flow, the process handler communicates state information to the request engine, such that the state and progress of the processing job at discrete steps is known by multiple processes, typically on different physical hive engines, and possibly in different territories (which may be defined to be in physically different locations, or using different communications and/or electrical systems, etc.) Thus, should a failure occur, the processing job typically can be resumed by another process handler newly selected by the request handler, or possibly completed by the original process handler with it storing results and/or communicating the results to the client via a different path (e.g., using a different request handler, etc.)

In one embodiment, processing a request typically includes the request setup, request processing, and request teardown. In the request setup, the client submits a request for a volunteer to the request region. A request handler receives the request, opens a TCP connection, and sends a response to the client. The client sends the request over the TCP connection to the request handler. The request handler receives the request and submits a request for a volunteer. A process handler receives the request, opens a TCP connection, and sends a response to the request handler. The request handler receives the response and sends the request over the TCP connection to the process handler. The process handler receives the request and sends an acknowledgement message. The request handler receives the acknowledgement message then sends an acknowledgement message to the client. The client receives the acknowledgement message then sends a process command to the request handler. The request handler receives the process command sends the process command to the process handler. The process handler receives the process command and begins processing the request. If the client loses connection with the request handler during this procedure, the client should perform a retry.

In one embodiment, in the request process procedure, the process handler submits a volunteer request to a processing region. A task handler receives the volunteer request, opens a TCP connection, and sends a response. The process handler receives the volunteer response and sends the first task in the process flow to the task handler over the TCP connection. The task handler processes the task and sends the results to the process handler. If the task does not complete within the specified amount of time and retries are set to zero, the request handler returns an error code as the final result to the request handler. If the task does not complete within the specified amount of time and retries are greater than zero, the request

handler resubmits the task to another task handler. If snapshot is enabled on this task or if retries is set to zero, the process handler sends the result to the request handler. This repeats until the next state is finish. When the next state is finish, the process handler sends the final result to the request handler. If the client loses connection with the request handler during this procedure, the client should perform a recover.

In one embodiment, in the request teardown procedure, the request handler sends the final result to the client. The client receives the result and sends an acknowledgement to the request handler. The request handler receives the acknowledgement and sends an acknowledgement to the process handler. If the client loses connection with the request handler during this procedure, the client should perform a recover.

In one embodiment, the task service runs on each worker machine. Task services have an IP address and assigned TCP port on their worker machine. All task services in the Hive share common UDP multicast groups based on their worker machine's current region. On completion of the volunteer pattern for a simple task, the connected TCP socket will be passed off to the task handler. When responding to a volunteer pattern for a daemon task, this service will UDP the daemon task's IP and port to the requester. The service has both task handlers and daemon tasks. Upon receiving a task to execute from a process handler, the service will spin off a task handler or delegate the task to a daemon task, as appropriate. Upon completion of the task, the task handler or daemon task will return the results to the process handler.

One embodiment uses an intra-process recovery which enables the hive to recover from a connection loss between the client and the request handler while the request handler is overseeing the processing of a request. When the client loses the connection with a first request handler, once the request processing has completed the request setup phase, the first request handler continues processing the request and the client submits a request for a new request handler (second request handler). The client issues the recover command and second request handler listens queries the recover service for a user-defined amount of time. If second request handler does not receive the result within the specified amount of time, second request handler returns an error. When first request handler receives the final result, first request handler writes the final result to the recover service.

One embodiment operates slightly differently as multiple process handlers are used for each step in a process flow. For example, both process handlers typically maintain the current state of the request such that if either of the process handlers is lost, the other picks up in its place. If the request handler is lost, the client and/or process handlers can establish a new request handler. The request handler manages the interface between software requesting processing from the hive and the hive. A primary process handler is a service that walks a request through the steps and recovery defined in a process flow. A secondary process handler is a service that monitors the primary process handler. If something happens to the primary process handler, the secondary process handler continues going through the steps and recovery defined in a process flow. A task handler is a service that performs the sub-routine defined in the process flow.

For example, in one embodiment, first, a request handler finds two process handlers. The request handler designates one as the primary process handler and the other as the secondary process handler. Next, the request handler sends the primary process handler the secondary process handler's IP address and sends the secondary process handler the primary process handler's IP address. The primary process handler and secondary process handler open a TCP port for communication then send acknowledgement messages to the request handler. The primary process handler finds a task handler. The task handler opens a TCP port and sends the request to the primary process handler. The primary process handler prepares the initial process flow state and sends that state to the secondary process handler. The secondary process handler and the request handler monitor the task states over the TCP connection. The task handler processes the request, sends the result to the primary process handler.

One embodiment provides an assimilation mechanism which recognizes new hive engines trying to join a hive. These steps occur without stopping execution of the entire hive, and he hive updates its hive engines in a measured rate to ensure that portions of the hive are continually processing requests ensuring constant availability of the hive applications.

In one embodiment, when a new hive engine joins the hive, the new hive engine finds the operating system image and the base hive software via DHCP. The new hive engine self installs the OS image and hive software using automated scripts defined by client. If a hive engine has an old version of the OS, the region leader makes the hive engine unavailable for processing. The hive engine is erased and rebooted. The hive engine then joins the hive as a new hive engine and re-installs the OS and hive software accordingly.

In addition, in one embodiment, when a hive engine joins the hive, the hive engine sends a request to the region leader. The hive engine receives a response from the region leader and selects a region to join. The region leader queries the hive engine for information about services, software, and versions. If the region leader is running a newer version of the hive system, the region leader makes the hive engine unavailable for processing. The region leader updates the hive engine by transmitting the current version of the hive system. The hive engine installs the update and commences processing. If the hive engine is running a newer version of hive system than the region leader, the region leader makes itself unavailable for process, receives the newer version of the hive system from the hive engine, installs the software, and continues processing. Once the region leader is updated, the region leader begins updating its region's members and the other region leaders. For example, in one embodiment, a hive engine then receives a response from the region leaders and selects a region to join. The region leader queries the hive engine for information about services, software, and versions. If the region leader is running the most current version of the hive applications, the region leader automatically updates the hive engine's hive applications. If the hive engine is running the most current version of the hive applications, the region leader automatically updates its hive applications. Once the region leader is updated, the region leader begins updating its region's members and the other region leaders.

Turning to the figures, FIG. **1**A illustrates an architecture of hives used in one embodiment. Shown are multiple hives **100-101**. A hive **100-101** is a logical grouping of one or more hive engines (e.g., computers or other computing devices) networked together to perform processing resources to one or more hive clients **110**. For example, hive **100** includes multiple hive engines **105-106** connected over a network (or any communication mechanism) **107**.

In one embodiment, a hive is a decentralized network of commodity hardware working cooperatively to provide vast computing power. A hive typically provides high-availability, high-scalability, low-maintenance, and predictable-time computations to applications (e.g., those corresponding to processing jobs of clients) executed in the hive. Each hive engine in the hive is typically capable to individually deploy and execute hive applications. When placed on the same network, hive engines seek each other out to pool resources and to add availability and scalability.

FIG. **1**B illustrates a computing platform used for a hive engine for implementing request handlers, process handlers, and/or other processes of a hive as used in one embodiment (or also used for simulating the operation of one or more elements of a hive in one embodiment). As shown, hive engine **120** is configured to execute request handlers, process handler, and other hive processes, and to communicate with clients and other hive engines as discussed herein.

In one embodiment, hive engine **120** includes a processing element **121**, memory **122**, storage devices **123**, communications/network interface **124**, and possibly resources/interfaces (i.e., to communicate to other resources) which may be required for a particular hive application (e.g., specialized hardware, databases, I/O devices, or any other device, etc.) Elements **121-125** are typically coupled via one or more communications mechanisms **129** (shown as a bus for illustrative purposes). Various embodiments of hive engine **120** may include more or less elements. The operation of hive engine **120** is typically controlled by processing element **121** using memory **122** and storage devices **123** to perform one or more hive processes, hive tasks, or other hive operations according to the invention. Memory **122** is one type of computer-readable medium, and typically comprises random access memory (RAM), read only memory (ROM), flash memory, integrated circuits, and/or other memory components. Memory **122** typically stores computer-executable instructions to be executed by processing element **121** and/or data which is manipulated by processing element **121** for implementing functionality in accordance with the invention. Storage devices **123** are another type of computer-readable medium, and typically comprise solid state storage media, disk drives, diskettes, networked services, tape drives, and other storage devices. Storage devices **123** typically store computer-executable instructions to be executed by processing element **121** and/or data which is manipulated by processing element **121** for implementing functionality in accordance with the invention.

In one embodiment, hive engine **120** is used as a simulation engine **120** to simulate one or more hive engines, and/or one or more hive processes, tasks, or other hive functions, such as, but not limited to those disclosed herein, especially the operations, methods, steps and communication of messages illustrated by the block and flow diagrams and messages sequence charts. Hive simulator engine **120** typically is used to simulate the performance and availability of hive application fabrics. The simulator allows dynamic simulation of any environment using simple text directives or a graphical user interface. For example, hive simulator engine **120** can be used to determine the hive performance using particular computing hardware by specifying such things as the computer type, instantiation parameters, and connection fabric, which is used by hive simulator engine **120** to produce a representation of the performance of a corresponding hive. In one embodiment, multiple hive simulator engines **120** are used, such as a unique three-level, two-dimensional mode connection fabric that allows hive simulator engines **120** to transmit requests uni-directionally or bi-directionally and to access other hive simulator engines **120** for subset processing while processing a request. Thus, one or more hive simulator engines **120** allow for modeling at the software level, hardware level, or both levels. Additionally, a hive simulator engine **120** is typically able to transmit requests through a simulated network or real hive network, such as hive **100** (FIG. **1**A).

FIG. **2**A illustrates a hierarchy of a hive, request regions, territories, and processing regions as used in one embodiment. As shown, hive **200** is logically divided into one or more request regions **205** (although most hives use only one request regions), territories **210** and **216**, with multiple processing regions **211-212** and **217-218**. The use of territories **210** and **216** provides a mechanism for associating a physical location or quality of a corresponding hive engine which can be used, for example, in determining which responding request or process handlers to select via a volunteer pattern. When defined based on physical location, if performance is the major issue, then it is typically advantageous (but not required) to process all requests within the same territory. If reliability is the major issue, then it is typically advantageous (but not required) store state recover information in another territory.

FIG. **2**B illustrates an interaction of a client, request handlers, and process handlers of one embodiment. Client **220** generates a service request **221** to request handlers **222**, such as via a request region multicast message, one or more messages, a broadcast message, or other communication mechanisms. Those request handlers **222** that are available to process the request return responses **223** to client **220**, typically via a unicast message directly to client **220** which includes a communications port to use should the sending request handler be selected by client **220**. Client **220** selects, optionally based on territory considerations, typically one (but possibly more) of the responding request handlers, and communicates processing job **224** to the selected request handler **225**.

In response, selected request handler **225** generates a processing request **226** to process handlers **227**, such a via one or more processing region multicast messages or other communication mechanisms. Those process handlers **227** that are available to process the request return responses **228** to selected request handler **225**, typically via a unicast message directly to selected request handler **225** which includes a communications port to use should the sending request handler be selected by selected request handler **225**. Selected request handler **225** selects, optionally based on territory considerations, typically one (but possibly more) of the responding process handlers, and communicates processing job with state information **229** to the selected process handler **230**. Inclusion of the state information is emphasized in regards to processing job with state information **229** because the processing job might be ran from the beginning or initialization state, or from an intermittent position or state, such as might happen in response to an error or timeout condition.

In response, selected process handler **230** proceeds to execute the process flow (or any other specified application), and at defined points in the process flow, updates selected request handler **225** with updated/progressive state information **237**. Typically based on the process flow, selected process handler **230** will sequentially (although one embodiment allows for multiple tasks or sub-processes to be executed in parallel) cause the tasks or processing requests to be performed within the same hive engine or by other hive engines.

In one embodiment, selected process handler **230** selects a hive engine to perform a particular task using a volunteer pattern. For example, selected process handler **230** sends a multicast task request **231** to task handlers typically within the processing region (although one embodiment, sends task requests **231** to hive engines in one or more processing and/or request regions). Those task handlers **232** able to perform the corresponding task send a response message **233** to selected process handler **230**, which selects, possibly based on territory, hive engine (e.g., itself as less overhead is incurred to perform the task within the same hive engine) or other considerations, one of the responding task handlers **232**. Selected process handler **230** then initiates the task and communicates state information via message **234** to the selected task handler **235**, which performs the task and returns state information **236** to selected process handler **230**. If there are more tasks to perform, selected process handler **230** typically then repeats this process such that tasks within a process flow or application may or may not be performed by different hive engines. Upon completion of the application/process flow, selected process handler **230** forwards the final state information (e.g., the result) **237** to selected request handler **225**, which in turn, forwards the result and/or other information **238** to client **220**.

In one embodiment, selected process handler **230** performs tasks itself or causes tasks to be performed within the hive engine in which it resides (and thus selected task handler **235** is within this hive engine, and one embodiment does not send task request message **231** or it is sent internally within the hive engine.) In one embodiment, selected task handler **235** is a separate process or thread running in the same hive engine as selected process handler **230**. Upon completion of the application/process flow, selected process handler **230** forwards the final state information (e.g., the result) **237** to selected request handler **225**, which in turn, forwards the result and/or other information **238** to client **220**.

FIG. **2**C illustrates multicast addresses **240** used in one embodiment. As shown, multicasts addresses **240** includes: a multicast request region address **241** using which a client typically sends a service request message, a processing region leader intercommunication multicast address **242** used for processing region leaders to communicate among themselves, a processing region active region indications multicast address **243** which is typically used to periodically send-out messages by region leaders to indicate which processing regions are currently active, and multiple processing region multicasts addresses **244**, one typically for each processing region of the hive. Of course, different sets or configurations of multicast addresses or even different communications mechanisms may be used in one embodiment within the scope and spirit of the invention.

FIG. **2**D illustrates the flow of messages among components of one embodiment. Client **250** sends a multicast hive service request message **256** into the request region **251** of the hive. Request handlers available for performing the application corresponding to request **256** respond with UDP messages **257** to client **250**, which selects selected request handler **252**, one of the responding request handlers. In one embodiment, this selection is performed based on territory or other considerations, or even on a random basis. Client **250** then communicates the processing job in a message **258** over a TCP connection to the selected request handler **252**.

In response and using a similar volunteer pattern, selected request handler **252** multicasts a processing request message **259** to a selected processing region **253**, and receives UDP response messages **260** from available processing engines to service the request (e.g., perform the processing job). Selected request handler **252** selects selected process handler **254**, one of the responding request handlers. In one embodiment, this selection is performed based on territory or other considerations, or even on a random basis. Selected request handler **252** then forwards the processing job with state information in message **261** to selected process handler **254**, which returns an acknowledgement message **262**. In response, selected request handler **252** sends an acknowledgement message **263** to client **250** (e.g., so that it knows that the processing is about to be performed.)

Selected process handler **254** then causes the processing job to be executed, typically by performing tasks within the same hive engine if possible for optimization reasons, or by sending out one or more tasks (possibly using a volunteer pattern) to other hive engines. Thus, selected process handler **254** optionally sends a multicast task request message **264** typically within its own processing region (i.e., selected processing region **253**) (and/or optionally to one or more other processing or request regions), and receives responses **265** indicating available task handlers for processing the corresponding task. Task request message **264** typically includes an indication of the type or name of the task or task processing to be performed so that task handlers/hive engines can use this information to determine whether they can perform the task, and if not, they typically do not send a response message **265** (as it is less overhead than sending a response message indicating the corresponding task handler/hive engine cannot perform the task.) Note, in one embodiment, a task handler within the same hive engine as selected process handler **254** sends a response message **265**.

Whether a task handler to perform the first task is explicitly or implicitly determined, selected process handler initiates a first task **266**, which is performed by one of one or more individual task threads **255** (which may be the same or different task threads on the same or different hive engines), which upon completion (whether naturally or because of an error or timeout condition), returns state information **272** to selected process handler **254**, which in turn updates selected request handler **252** via progressive state message **273**. (Note, if there was only one task, then completion/state message **276** would have been sent in response to completion of the task.) This may continue for multiple tasks as indicated by optional MCAST task request and response messages **268**-**269** and task-n initiation **270** and state messages **272**. When processing of the application/process flow is completed as determined by selected process handler **254** in response to state messages from the individual task threads **255**, selected process handler **254** forwards a completion and result state information **276** to selected process handler **252**, which forwards a result message **277** to client **250**. In response, client **250** sends an acknowledgement message **278** to confirm receipt of the result (indicating error recovery operations do not need to be performed), and an acknowledgement message **279** is forwarded to selected process handler **254**, and processing of the processing job is complete.

FIG. **2**E illustrates an interaction of a client, request handlers, process handlers and possibly tasks of one embodiment. Many of the processes and much of the flow of information is the same as illustrated in FIG. **2**B and described herein, and thus will not be repeated. FIG. **2**E is used to emphasize and explicitly illustrate that different embodiments may implement features differently, and to emphasize that a process flow may specify tasks or even other process flows to be performed or the same process flow to be performed recursively.

For example, as shown, selected process handler **230** of FIG. **2**B is replaced with selected process handler **280** in FIG. **2**E. Selected process handler **280**, in response to being assigned to execute the clients processing job by receiving processing job with state information message **229**, proceeds to execute the corresponding application/process flow, which may optionally include performing a volunteer pattern using processing or task request messages **281** and response messages **283** to/from one or more task or process handlers **282**. In response to the volunteer operation or directly in response to receiving the processing job with state information message **229**, selected process handler **280** will sequentially (although one embodiment allows for multiple tasks or subprocesses to be executed in parallel) perform itself or send out

tasks or processing requests to corresponding selected task or process handlers **290**, in which case task or processing job with state information messages **284** are typically sent and results or state information messages **296** are typically received. The number of levels used in performing a processing job is unbounded as indicated in FIG. **2**E.

FIG. **3** is a flow diagram of a client process used in one embodiment. Processing begins with process block **300**, and proceeds to process block **302**, wherein an application, data, and hive to process these is identified. Next, in process block **304**, a multicast service request message indicating application is sent into the request layer of the selected hive. In process block **306**, responses are received from the hive (if no responses are received, processing returns to process block **302** or **304** in one embodiment). Next, in process block **308**, a request handler is selected based on the responses, and a communications connection is established to the selected request handler in process block **310**. Next, in process block **312**, the processing job is submitted to the selected request handler and a global unique identifier (GUID) is included so that the client and hive can uniquely identify the particular processing job. As determined in process block **314**, if an acknowledgement message is not received from the hive indicating the job is being processed within a timeframe, then processing returns to process block **304**.

Otherwise, if results are received from the hive within the requisite timeframe as determined in process block **320**, then an acknowledgement message is returned to the hive in process block **322**, and processing is complete as indicated by process block **324**. Otherwise, as determined in process block **330**, if the client determines it wishes to perform a recover operation, then in process block **332**, a multicast recovery request message specifying the GUID is sent to the request layer of the hive, and processing returns to process block **320** to await the recovery results. Otherwise, as determined in process block **340**, if the client determines to again request the job be performed, then processing returns to process block **304**. Otherwise, local error processing is optionally performed in process block **342**, and processing is complete as indicated by process block **344**.

FIGS. **4**A-C are flow diagrams of request hander processes used in one embodiment. FIG. **4**A illustrates a process used in one embodiment for responding to service requests of clients. Processing begins with process block **400**, and proceeds to process block **402**, wherein a multicast port is opened for receiving service request messages. As determined in process blocks **404** and **406**, until a service request is received and the request handler is available to handle the request, processing returns to process block **404**. Otherwise, the request handler responds in process block **408** by sending a response message to the requesting client, with the response message typically identifying a port to use and the GUID of the received service request. As determined in process block **410**, if the service request corresponds to a recovery request, then in process block **412**, a recovery thread is initialized (such as that corresponding to the flow diagram of FIG. **4**C) or the recovery operation is directly performed. Otherwise, in process block **414**, a selected request handler thread is initialized (such as that corresponding to the flow diagram of FIG. **4**B) or the request is handled directly. Processing returns to process block **404** to respond to more requests.

FIG. **4**B illustrates a flow diagram of a process used by a selected request handler in one embodiment. Processing begins with process block **430**, and loops between process blocks **432** and **434** until a job is received (and then processing proceeds to process block **440**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **436**.

After a processing job has been received (e.g., this process has been selected by the client to handle the request), a state data structure is initialized in process block **440**. Then, in process block **442**, a multicast processing request message is sent into one of the processing layers of the hive. As determined in process block **444**, if no responses are received within a requisite timeframe, then a no processing handler response message is returned to the client in process block **445**, and processing is complete as indicated by process block **436**.

Otherwise, in process block **446**, a particular process handler is selected. In one embodiment, this selection is performed based on territories (e.g., a process handler in a different territory than the selected request handler), other considerations or even on a random basis. In process block **448**, a communications connection is established if necessary to the selected process handler, and the state information and data for the client processing request is sent (which may correspond to the initial state of the data received from the client or to an intermediate state of processing the client job request).

As determined in process block **450**, if an error or timeout condition is detected, processing returns to process block **442**. Otherwise, as determined in process block **452**, until a state update message is received, processing returns to process block **450**. As determined in process block **454**, if the received state is not the finished or completed state, then in process block **456**, the state data structure is updated, and processing returns to process block **450**. Otherwise, processing has been completed, and in process block **458**, the result is communicated to the client; in process block **460**, the communications connection is closed; and processing is complete as indicated by process block **462**.

FIG. **4**C illustrates a flow diagram of a process used by a selected request handler performing error recovery in one embodiment. Processing begins with process block **470**, and loops between process blocks **472** and **474** until a job is received (and then processing proceeds to process block **478**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **476**.

After a processing job has been received (e.g., this process has been selected by the client to perform the recover processing), in process block **478**, a multicast recovery request message specifying the GUID of the job being recovered is sent into one or more of the recovery modules of the hive. As determined in process block **480**, if no responses are received within a requisite timeframe, then a no recover response message is returned to the client in process block **481**, and processing is complete as indicated by process block **476**.

Otherwise, in process block **482**, a particular recovery handler is selected, possibly based on territory considerations—such as a recovery handler in a different territory then this selected request handler. In process block **484**, a communications connection is established if necessary to the selected recovery handler thread, and a recovery request is sent, typically including the GUID or other indication of the job to be recovered.

As determined in process block **486**, if an error or timeout condition is detected, processing returns to process block **478**. Otherwise, the recovered information is received as indicated by process block **488**. In process block **490**, the information is typically communicated to the client, or if this communication fails, it is saved to the recovery system. In one embodiment, the partially completed state, errors and/or other indications are stored to a local storage mechanism (e.g., some computer-readable medium) to be made available for use by a recovery process. In one embodiment, more significant process handling is performed, or the error communicating the error to another process, thread or hive engine for handling. The communications connection is then closed in process block **492**, and processing is complete as indicated by process block **494**.

FIGS. **5**A-B are flow diagrams of process hander processes used in one embodiment. FIG. **5**A illustrates a process used in one embodiment for responding to service requests of request handlers. Processing begins with process block **500**, and proceeds to process block **502**, wherein a multicast port is opened for receiving processing request messages. As determined in process blocks **504** and **506**, until a processing request is received and the process handler is available to handle the request, processing returns to process block **504**. Otherwise, the process handler responds in process block **508** by sending a response message to the requesting request handler, with the response message typically identifying a port to use and possibly the GUID corresponding to the received processing request. The processing request is received in process block **510**. Next, in process block **512**, a selected process handler thread is initialized (such as that corresponding to the flow diagram of FIG. **5**B) or the processing request is handled directly. Processing returns to process block **504** to respond to more requests.

FIG. **5**B illustrates a flow diagram of a process used by a selected process handler in one embodiment. Processing begins with process block **520**, and loops between process blocks **522** and **524** until a job is received (and then processing proceeds to process block **530**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **526**.

After a processing job has been received (e.g., this process has been selected by a selected request handler (or possibly other process handler) to handle the request), a state data structure is initialized in process block **530**. In process block **532**, the processing requirements of the next statement(s) within the process flow corresponding to the received job are identified. As determined in process block **534**, if a sub-process is to be spawned (e.g., the process flow specifies a process flow to be executed), then in process block **536**, the current state is pushed on to a state stack and the state is initialized to that of the new process flow, the selected request handler is updated in process block **538**, and processing returns to process block **532** to process the new process flow.

Otherwise, as determined in process block **540**, if the task handler is not already known (e.g., an optimization to perform the task on the same hive engine) such as it is not guaranteed to be performed locally, the task is a "limited task" in that it can only be performed by a subset of the task handlers or the processing of the task is made available to other hive engines (e.g., for performance or load balancing etc.), then in process block **542** the task handler to perform the task is identified. One embodiment identifies the task handler by sending a multicast task request messages, receives the responses, and selects, based on territory, load or other considerations, a task handler to perform the task.

Limited tasks provide a mechanism for identifying hive engines that have special hardware or other resources. Task handlers only on the hive engines with the specialized hardware or other resources possibly required to perform the task will be enabled to perform the corresponding task and thus these enabled task handlers will be the ones to respond to a task request for the corresponding task. Additionally, limited tasks provide a mechanism to limit the number of task handlers or hive engines allowed to access a particular resource by restricting the number and/or location of task handlers allowed to perform a task that accesses the particular resource. Thus, limited tasks may be useful to limit the rate or number of accesses to a particular resource (e.g., database engine, a storage device, a printer, etc.)

In process block **544**, a task is initiated to perform the next operation identified in the current process flow with the current state information and characteristics (e.g., timeout, number of retries, etc.) on the identified, selected, or already known task handler. As determined in process block **546**, after completion of the processing requirements of the processing statement(s), if the finish state has not been reached, then the state data structure is updated with the task result in process block **548**, the selected request handler is updated with the current state information in process block **549**, and processing returns to process block **532**.

Otherwise, processing is completed of the current process flow as determined in process block **546**, and if the current process flow is a sub-process (e.g., spawned process flow) (as determined in process block **550**), then in process block **552**, the state is popped from the state stack, and processing proceeds to process block **548**. Otherwise, in process block **554**, the result/state information is communicated to the selected request hander. As determined in process block **555**, if an error has been detected, then error processing is performed in process block **556**. In process block **558**, the communications connection is closed, and processing is complete as indicated by process block **559**. Note, in some embodiments, communications connections are not established and disconnected each time, but rather a same communications channel is used more than once.

FIG. **5**C illustrates a flow diagram of a task handler performed by a hive engine in one embodiment. Processing begins with process block **580**. As determined in process blocks **581** and **583**, until a task request is received and the task handler is available to handle the request, processing returns to process block **581**. Otherwise, the task handler responds in process block **584** by sending a response message to the requesting process (typically a process handler), with the response message typically identifying a port to use and the GUID of the received task request. As determined in process block **585**, if the task is actually received (e.g., this task handler was selected by the process handler sending the task request), then in process block **586**, the task is performed or at least attempted to be performed and resultant state information (e.g., completed state, partially completed state, errors and/or other indications) sent to the requesting process handler. Processing returns to process block **581**. Note, in one embodiment, multiple processes illustrated in process block **5**C or some variant thereof are performed simultaneously by a hive engine for responding to multiple task requests and/or performing tasks in parallel.

FIG. **5**D illustrates a flow diagram of a recovery processing performed by a hive engine in one embodiment. Processing begins with process block **590**, and loops between process blocks **591** and **592** until a recovery job is received (and then processing proceeds to process block **594**) or until a timeout condition is detected and in which case, processing is complete as indicated by process block **593**. In process block **594**, the recovery is retrieved from local storage and is communicated to the selected request hander. As determined in process block **595**, if an error has been detected, then error processing is performed in process block **595**. In process block **598**, the communications connection is closed, and processing is complete as indicated by process block **599**.

In one embodiment, a hive application is a collection of process flows that carry out specific sets of tasks. Applica-

tions can share process flows. An application definition file (XML descriptor file) typically describes the application, and the application definition file typically consists of the following: application name, process flow names, task names and module file names, support files, and/or configuration file names.

FIG. **6**A illustrates an example definition file **600** of an application for use in one embodiment. As show, application definition file **600** specifies a set of corresponding process flows **601**, tasks **602**, support files **603**, and configuration files **604**.

FIG. **6**B illustrates a definition of an process flow **620** "doProcessOne" used in one embodiment. Shown are four process flow statements **621**-**624**, each specifying its beginning state, tasks to be performed, and next state depending on the outcome of the statements execution.

FIG. **6**C illustrates a process used in one embodiment for executing a process flow or processing job, such as that illustrated in FIG. **6**B. Note, in one embodiment, the process illustrated in FIG. **5**B is used to execute a process flow or processing job. In one embodiment, a combination of the processes illustrated in FIGS. **5**B and **6**C or another process is used to execute a process flow or processing job.

Turning to FIG. **6**C, processing begins with process block **650**, and proceeds to process block **652**, wherein the current state is set to the START state. Next, in process block **654**, the task associated with the current state is attempted to be performed. As determined in process block **656**, if the task timed-out before completion, then as determined in process block **658**, if the task should be retried (e.g., the number of retries specified in the process flow or a default value has not been exhausted), processing returns to process block **656**. Otherwise, in process block **660**, the current state is updated to that corresponding to the task's completion status (e.g., complete, non-complete, not-attempted, etc.). As determined in process block **662**, if an error occurred (e.g., an invalid next state or other error condition), then an error indication is returned to the selected request handler in process block **664**, and processing is complete as indicated by process block **666**. Otherwise, if the next state is the FINISH state (as determined in process block **670**), then the result and possibly a final set of state information is sent to the selected request handler in process block **672**, and processing is complete as indicated by process block **672**. Otherwise, in process block **674**, the selected request handler is updated with current state information, such as, but not limited to (nor required to include) the current state name, intermediate results, variable values, etc. Processing then returns to process block **654**.

One embodiment of a hive uses a logical hierarchy of hive engines for delegation of performing administrative and/or other hive related tasks. In one embodiment, each hive engine participates in the processing region hierarchy as a region member with one hive engine in each processing region being a region leader, and there one overall senior region leader for the hive. For example, shown in FIG. **7**A are multiple processing regions **700**-**701**, having an overall senior region leader **703** (denoted senior leader/region leader/region member as it performs all functions) residing in processing region **700**, a region leader/region member **707** in processing region **701**, region members **704**-**705** in processing region **700**, and region members **708**-**709** in processing region **701**.

FIGS. **7**B-**7**C are flow diagrams of processes used in one embodiment to establish and maintain this hierarchical relationship among distributed processes or systems, such as among hive engines. The generic terms of heartbeat leader and heartbeat member are used in describing this process, because it can be used in many different applications for establishing and maintaining a hierarchical relationship in a set of dynamic and autonomous processes and systems. For example, in one embodiment, the processes illustrated in FIGS. **7**B-C are used to establish and maintain which hive engine in a region is the region leader, and between region leaders for establishing which hive engine is the senior region leader.

Processing of the heartbeat leader flow diagram illustrated in FIG. **7**B begins with process block **720**, and proceeds to process block **722** wherein a multicast heartbeat request message is sent on the multicast address belonging to the group in which the hierarchical relationship is being established and maintained. In process block **724**, the responses are received. As determined in process block **725**, if the process is senior over those from which a response was received, then it remains the leader or senior process, and optionally in process block **726**, piggybacked information (e.g., number of regions, number of members in each region, etc.) is processed and possibly actions taken or initiated in response. As indicated by process block **727**, the process delays or waits a certain period of time before repeating this process, and then processing returns to process block **722**. Otherwise, in process block **728**, the process demotes itself from being the leader or senior process (such as by initiating or switching to performing actions consistent with being a region member if not already performing the functions of a region member), and processing is complete as indicated by process block **729**.

Processing of the heartbeat member flow diagram illustrated in FIG. **7**C begins with process block **740**, and proceeds to process block **742**, wherein the process watches for and identifies heartbeat request messages during a predetermined timeframe. As determined in process block **744**, if a no heartbeat request is received, then in process block **745**, the process promotes itself to being the heartbeat leader, and processing returns to process block **742**. Otherwise, if this process is senior to a process sending a heartbeat request message as determined in process block **748**, then processing proceeds to process block **745** to promotes itself. Otherwise, in process block **749**, a heartbeat response message is sent to the sender of the received heartbeat request message, and optionally other information is included in the heartbeat response message. Processing then returns to process block **742**. Note, determining seniority can be performed in numerous manners and mechanisms, such as that based on some physical or logical value associated with a hive engine (e.g., one of its network addresses, its serial number, etc.)

FIG. **8**A illustrates some of the functions performed by a senior processing region leader in one embodiment. Processing begins with process block **800**, and proceeds to process block **802**, wherein a heartbeat request is sent to all region leaders, typically by sending a multicast packet to the processing region leader intercommunication multicast address **242** (FIG. **2**C) and piggybacked information is collected from received responses with this information typically including, but not limited to the number of processing regions, number of processing handlers, number of request handlers, limited task information, etc. As determined in process block **804**, if the number of request handlers needs to be adjusted (e.g., there are too few or too many), then in process block **806**, a region leader is selected and directed to start or stop a request handler. Next, as determined in process block **808**, if the number of processing regions needs to be adjusted (e.g., there are too few or too many), then in process block **810**, a region leader is selected and directed to disband or spit a region. Next, as determined in process block **812**, if the number of task handlers that can perform a particular task (i.e., a "limited task" as typically and by default, all tasks can be per-

formed by all task handlers) needs to be adjusted (e.g., there are too few or too many), then in process block **814**, a region leader is selected and directed to adjust the number of task handlers within its region which can perform the particular limited task. Next, as determined in process block **816**, if some other action needs to be performed, then in process block **818**, the action is performed or a region leader is instructed to perform the action. Next, processing usually waits or delays for a predetermined or dynamic amount of time as indicated by process block **819**, before processing returns to process block **802**.

FIG. **8**B illustrates some of the functions performed by a region leader in one embodiment. Processing begins with process block **830**, and proceeds to process block **832**, wherein a heartbeat request is sent to all region member, typically by sending a multicast packet to the processing region multicast address **244** (FIG. **2**C), and piggybacked information is collected from received responses with this information typically including, but not limited to the number of processing handlers, number of request handlers, etc.; or possibly instructions are received from the senior region leader.

As determined in process block **834**, if the number of request handlers needs to be adjusted (e.g., there are too few or too many), then in process block **836**, a process handler is selected and directed to start or stop a request handler. Next, as determined in process block **838**, if the number of processing regions needs to be adjusted (e.g., there are too few or too many), then in process block **840**, an instruction to disband or spit the region is issued. Next, as determined in process block **842**, if the number of task handlers permitted to perform a particular limited task needs to be adjusted (e.g., there are too few or too many), then in process block **844**, an instruction is provided (directly, indirectly such as via a request or process handler, or based on a volunteer pattern) to a particular task handler to permit or deny it from performing the particular limited task. Next, as determined in process block **846**, if some other action needs to be performed, then in process block **848**, the action is performed or a process handler is instructed to perform the action. Next, processing usually waits or delays for a predetermined or dynamic amount of time as indicated by process block **849**, before processing returns to process block **832**.

FIG. **8**C illustrates the splitting of a region as performed in one embodiment. Region leader **860** sends a multicast message **871** requesting a volunteer to head the new region to region members **861**, some of which typically return a positive response message **872**. Region leader **860** then identifies a selected region member **862** to head the new processing region, and sends an appointment message **873** to selected region member **862**. In response, selected region member **862** creates a new processing region as indicated by reference number **874**, typically including identifying an unused processing region multicast address **244** (FIG. **2**C) as it monitored the traffic or processing region active indication messages sent to processing region active region indications multicast address **243**. Then, selected region member **862** multicasts a volunteer message **875** to processing regions in the old (and still used) processing region and typically receives one or more responses **876**. Selected region member **862** then selects a certain number, typically half of the number of process handlers in the old processing region, of responding process handlers, and notifies them to switch to the new processing region via move instruction **877**, and they in turn, send a confirmation message **878** to selected region member **862**.

FIG. **9** illustrates a process used in one embodiment for initializing a hive engine. Processing begins with process block **900**, and proceeds to process block **902**, wherein a hive version request and hive join multicast message is sent typically to all region leaders. As determined in process block **904**, if no responses are received, then in process block **912**, a new processing region is formed, and request hander and region leader processes are initiated. Next, in process block **914**, process handler, recovery module, and region member processes are initiated, and startup processing is completed as indicated by process block **916**. Otherwise, as determined in process block **906**, if a hive software update is available, then, in process block **908**, one of the responders is selected, the updates are acquired, and the software (e.g., hive software, operating system, etc.) is updated. In process block **910**, the hive engine joins the smallest or possibly one of the smaller processing regions, possibly with this selection being determined by identified territories, and processing proceeds to process block **914**.

In one embodiment, the hive is updated by a client with special administrative privileges. This administrative client sends a request to the senior region leader of the hive. The senior region leader opens a TCP connection and sends the administration client the connection information. The administration client sends the new application to the senior region leader. When the senior region leader receives an update, the senior region leader multicasts the update command to all the hive members. The senior region leader sends multicast message containing the name of the file that is being updated, the new version, and the total number of packets each hive member should receive. The senior region leader then multicasts the data packets, each packet typically includes the file id, the packet number, and data. If a hive member does not receive a packet, that hive member sends a request to the senior region leader for the missing packet. The senior region leader resends, multicasts, the missing packet. The hive members store the update in a staging area until they receive the activation command. To activate an update, the administration client sends the activation command to the senior region leader. The senior region leader multicasts the activate command to the hive members. The hive members remove the old application or files and moves the update from the staging area to the production area. To update the hive software or operating system, the senior region leader distributes the updates and restarts volunteers in a rolling fashion. When the hive service manager detects a new version of itself, the service manager forks the process and restarts with a new version. Also, the senior region leader can send other update commands. An active message indicates that the corresponding application, patch, or OS that should be running on the hive. A deactivated messages indicates that the corresponding application, patch, or OS should not be running on the hive and should remain installed on hive members. A remove message indicates that the corresponding application, patch, or OS was once installed on the Hive and any instances found on Hive members should be removed. This allows hive engines to be updated and also to move back to previous releases.

In view of the many possible embodiments to which the principles of our invention may be applied, it will be appreciated that the embodiments and aspects thereof described herein with respect to the drawings/figures are only illustrative and should not be taken as limiting the scope of the invention. For example and as would be apparent to one skilled in the art, many of the process block operations can be re-ordered to be performed before, after, or substantially concurrent with other operations. Also, many different forms of

data structures could be used in various embodiments. The invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

What is claimed is:

1. A system for processing information, the system comprising:

a plurality of networked computers for processing a plurality of processing jobs in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow;

the request handler configured to (1) receive a plurality of service requests for the processing jobs, (2) store state information for the processing jobs, and (3) communicate data relating to the processing jobs to a plurality of the process handlers;

the process handlers to which the data relating to the processing jobs were communicated being configured to (1) analyze the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identify a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determine that the processing job corresponding to the process flow with no remaining processing tasks has been completed; and

the task handlers configured to perform the identified processing tasks to generate a plurality of task results; and

wherein the request handler is further configured to store updated state information for the processing jobs, the updated stored state information being based on the task results.

2. The system of claim **1** wherein the process handlers are further configured to volunteer for servicing the processing jobs based on their availabilities for servicing the processing jobs.

3. The system of claim **2** wherein the request handler is further configured to (1) select the process handlers for servicing the processing jobs from among the process handlers that volunteered for same, and (2) communicate the data relating the processing jobs to the selected process handlers.

4. The system of claim **3** wherein the selected process handlers are further configured to communicate a plurality of task requests for the identified processing tasks to a plurality of the task handlers;

wherein the tasks handlers to which the task requests were communicated are further configured to volunteer for performing the identified tasks corresponding to the task requests based on their being able to perform the identified tasks corresponding to the task requests; and

wherein the selected process handlers are further configured to select the task handlers for performing the identified tasks corresponding to the task requests from among the task handlers that volunteered for same.

5. The system of claim **3** wherein the task handlers are further configured to communicate updated state information for the processing jobs following completion of their identified processing tasks to the selected process handlers.

6. The system of claim **5** wherein the selected process handlers are further configured to communicate updated state information for the processing jobs to the request handler.

7. The system of claim **2** wherein the request handler is further configured to communicate a plurality of processing requests for the processing jobs to a plurality of the process handlers;

wherein the process handlers to which the processing requests were communicated are further configured to volunteer for servicing the processing jobs corresponding to the processing requests based on their availabilities for servicing the processing jobs corresponding to the processing requests; and

wherein the request handler is further configured to (1) select the process handlers for servicing the processing jobs corresponding to the processing requests from among the process handlers that volunteered for same, and (2) communicate the data relating to the processing jobs to the selected process handlers.

8. The system of claim **1** wherein the task handlers are further configured to volunteer for performing the identified tasks based on their being able to perform the identified tasks.

9. The system of claim **8** wherein a plurality of the networked computers on which a plurality of the task handlers reside have different resources; and

wherein a plurality of the task handlers are further configured to volunteer for performing the identified tasks as a function of the resources of the networked computers on which the plurality of task handers are resident.

10. The system of claim **9** wherein the different resources comprise different specialized hardware.

11. The system of claim **8** wherein a plurality of the networked computers on which a plurality of the task handlers are resident are grouped into a plurality of territories, and wherein the system is configured to assign at least one of the identified tasks to a task handler based at least in part on territory considerations.

12. The system of claim **11** wherein a plurality of the networked computers have an associated physical location, and wherein the plurality of the networked computers having an associated physical location are grouped into the territories based on their associated physical locations such that each territory comprises a plurality of networked computers having an associated physical location within that territory.

13. The system of claim **1** wherein a plurality of the networked computers on which a plurality of the task handlers reside have different resources; and

wherein the system is configured to assign a plurality of the identified tasks to a plurality of the task handlers based at least in part on the resources of the networked computers on which the plurality of task handers are resident.

14. The system of claim **13** wherein the system is further configured to assign at least one of the identified tasks to a task handler to a task handler in order to limit a number of accesses to a database engine that will be needed to perform the at least one assigned task.

15. The system of claim **13** wherein the system is further configured to assign at least one of the identified tasks to a task handler in order to limit a rate of access to a database engine that will be needed to perform the at least one assigned task.

16. The system of claim **13** wherein the system is further configured to assign at least one of the identified tasks to a task handler in order to limit a number of accesses to a storage device that will be needed to perform the at least one assigned task.

17. The system of claim **13** wherein the system is further configured to assign at least one of the identified tasks to a task handler in order to limit a rate of access to a storage device that will be needed to perform the at least one assigned task.

18. The system of claim **13** wherein the different resources comprise different specialized hardware.

19. The system of claim **13** wherein the assigned tasks comprise limited tasks.

20. The system of claim **13** wherein at least one of the networked computers is further configured to select a task handler to which an identified task will be assigned from among a plurality of the task handlers based at least in part on the resources of the networked computers on which the a plurality of the task handers are resident.

21. The system of claim **20** wherein the at least one networked computer comprises a process handler to which data relating to a processing job was communicated.

22. The system of claim **21** wherein the at least one of the networked computers has a process handler and a task handler resident therein.

23. The system of claim **13** wherein at least one of the networked computers is further configured to select a task handler to which an identified task will be assigned from among a plurality of the task handlers based at least in part on load balancing considerations.

24. The system of claim **1** wherein a plurality of the networked computers on which a plurality of the task handlers are resident are grouped into a plurality of territories, and wherein the system is configured to assign at least one of the identified tasks to a task handler based at least in part on territory considerations.

25. The system of claim **24** wherein a plurality of the networked computers have an associated physical location, and wherein the plurality of the networked computers having an associated physical location are grouped into the territories based on their associated physical locations such that each territory comprises a plurality of networked computers having an associated physical location within that territory.

26. The system of claim **1** wherein the request handler is further configured to (1) receive at least one of the service requests from a client computer, and (2) communicate a processing result for the processing job corresponding to the at least one service request to the client computer.

27. The system of claim **26** further comprising the client computer, the client computer in communication via a network with the networked computer on which the request handler is resident.

28. The system of claim **1** wherein the request handler is resident on a first of the networked computers, wherein at least one of the process handlers is resident on a second of the networked computers, and wherein at least one of the task handlers is resident on a third of the networked computers.

29. The system of claim **1** wherein the request handler and at least one of the process handlers are resident on the same one of the networked computers.

30. The system of claim **1** wherein the request handler is resident on a different one of the networked computers than the networked computers on which the process handlers and task handlers are resident.

31. The system of claim **30** wherein the process handlers are resident on different ones of the networked computers than the networked computers on which the task handlers are resident.

32. The system of claim **1** wherein at least one of the process handlers and at least one of the task handlers are resident on the same one of the networked computers, the at least one task handler being on a different thread of the networked computer than the at least one process handler.

33. The system of claim **1** wherein the logic for a process flow comprises (1) a plurality of state variables before and after each processing task of that process flow, the state variables configured to store the state information, and (2) a plurality of transitions from state to state based on results for the processing tasks of that process flow.

34. The system of claim **33** wherein the networked computers are configured to process the processing jobs according to a plurality of process definition files corresponding to the processing jobs, each process definition file being configured to define the process flow for a processing job.

35. The system of claim **1** wherein at least one of the process flows further includes a recovery procedure for the processing tasks of the at least one process flow.

36. The system of claim **1** wherein the task results include updated state information for the processing jobs; and

wherein the request handler is further configured to store the updated state information based on the updated state information included in the task results.

37. The system of claim **4** wherein the selected process handlers are further configured to communicate the task requests to the task handlers by sending the task requests to the task handlers.

38. The system of claim **2** wherein the tasks handlers are further configured to volunteer for performing the identified tasks based on their being able to perform the identified tasks.

39. The system of claim **38** wherein a plurality of the process handlers are further configured to communicate a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

40. The system of claim **2** wherein the request handler is further configured to assign global unique identifiers (GUIDs) to the processing jobs.

41. The system of claim **40** wherein the request handler is further configured to (1) receive at least one of the processing job service requests from a client computer, and (2) provide the client computer with the GUID assigned to that processing job.

42. The system of claim **40** wherein the request handler is further configured to (1) determine whether a recovery procedure is to be initiated for a processing job, and (2) in response to a determination that a recovery procedure is to be initiated for a processing job, generate a recovery request message that includes the GUID for that processing job.

43. The system of claim **42** wherein the networked computers further comprise a recovery handler, and wherein the request handler is further configured to communicate the generated recovery request message to the recovery handler.

44. The system of claim **1** wherein the processing jobs define a plurality of transactions.

45. The system of claim **44** wherein the transactions comprise a plurality of independent transactions.

46. The system of claim **44** wherein the transactions comprise a plurality of credit card processing transactions.

47. The system of claim **2** wherein each processing job is associated with a data structure that defines the process flow, the data structure including data that identifies a plurality of files for carrying out tasks, each file associated with a task.

48. The system of claim **47** wherein each of a plurality of the process flow data structures further comprises:

a plurality of state identifiers for different states of the process flow;

a plurality of task identifiers for different tasks of the process flow, each task identifier being associated with a state identifier; and

a plurality of associations between the state identifiers and a plurality of potential outcomes of the tasks corresponding to the task identifiers.

49. The system of claim **48** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of retry count limits, each retry count limit data being associated with a task identifier to define how many retries are to be performed for the associated task before an error condition is found.

50. The system of claim **48** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of timeout count limits, each timeout count limit data being associated with a task identifier to define how many timeouts are to be permitted for the associated task before an error condition is found.

51. The system of claim **47** wherein the tasks handlers are further configured to volunteer for performing the identified tasks based on their being able to perform the identified tasks; and

wherein a plurality of the process handlers are further configured to communicate a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

52. A method for processing information, the method comprising:

receiving a service request for a processing job, the processing job having an associated process flow, the process flow including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the process flow; and

executing the processing job in a distributed manner by a plurality of networked computers and in accordance with the received service request, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, wherein the executing step comprises:

the request handler storing state information for the processing job;

the request handler communicating data for the processing job to a process handler;

the process handler to which the data for the processing job was communicated (1) analyzing the state information for the processing job to determine whether any processing task in the process flow remains to be performed based on the logic for the process flow, (2) in response to the state information analysis indicating that a processing task remains for the process flow, identifying a processing task to be performed, and (3) in response to the state information analysis indicating that no processing task remains for the process flow, determining that the processing job has been completed;

the task handlers performing the identified processing tasks to generate a plurality of task results; and

updating the stored state information based on the task results.

53. The method of claim **52** wherein the executing step further comprises:

the task handlers volunteering to perform the identified tasks based on their being able to perform the identified tasks.

54. The method of claim **53** wherein the executing step further comprises:

the process handler to which the data for the processing job was communicated selecting the task handlers for performing the identified tasks in response to the volunteering.

55. The method of claim **52** wherein the receiving step comprises receiving a plurality of service requests for a plurality of the processing jobs;

wherein the executing step further comprises the networked computers executing the processing jobs in a distributed manner in accordance with the received service requests; and

wherein the executing step further comprises the process handlers volunteering for servicing the processing jobs based on their availabilities for servicing the processing jobs.

56. The method of claim **55** wherein the executing step further comprises:

the task handlers volunteering to perform the identified tasks for the processing jobs based on their being able to perform the identified tasks.

57. The method of claim wherein the processing jobs define a plurality of transactions.

58. The system of claim **57** wherein the transactions comprise a plurality of independent transactions.

59. The system of claim **57** wherein the transactions comprise a plurality of credit card processing transactions.

60. The method of claim **53** wherein the executing step further comprises:

a process handler among the plurality of process handlers volunteering for servicing the processing job based on its availability for servicing the processing job; and

wherein the communicating step comprises the request handler communicating data for the processing job to the process handler that volunteered for servicing the processing job.

61. The method of claim **53** wherein the executing step further comprises:

the process handler to which the data for the processing job was communicated communicating a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

62. The method of claim **52** further comprising:

the request handler assigning a global unique identifier (GUID) to the processing job.

63. The method of claim **62** wherein the receiving step comprises the request handler receiving the service request for the processing job from a client computer, the method further comprising:

the request handler providing the client computer with the GUID assigned to the processing job.

64. The method of claim **62** wherein the executing step further comprises:

the request handler determining whether a recovery procedure is to be initiated for the processing job; and

in response to a determination that the recovery procedure is to be initiated for the processing job, the request handler generating a recovery request message that includes the GUID for the processing job.

65. The method of claim **64** wherein the networked computers further comprise a recovery handler, and wherein executing step further comprises:

the request handler communicating the generated recovery request message to the recovery handler.

66. The method of claim **52** wherein the processing job is associated with a data structure that defines the process flow, the data structure including data that identifies a plurality of files for carrying out tasks, each file associated with a task.

67. The method of claim **66** wherein the process flow data structure further comprises:

a plurality of state identifiers for different states of the process flow;

a plurality of task identifiers for different tasks of the process flow, each task identifier being associated with a state identifier; and

a plurality of associations between the state identifiers and a plurality of potential outcomes of the tasks corresponding to the task identifiers.

68. The method of claim **67** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of retry count limits, each retry count limit data being associated with a task identifier to define how many retries are to be performed for the associated task before an error condition is found.

69. The method of claim **67** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of timeout count limits, each timeout count limit data being associated with a task identifier to define how many timeouts are to be permitted for the associated task before an error condition is found.

70. The method of claim **66** wherein the executing step further comprises:

the task handlers volunteering to perform the identified tasks based on their being able to perform the identified tasks; and

the process handler to which the data for the processing job was communicated communicating a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

71. The method of claim **52** wherein the executing step further comprises:

a process handler among the plurality of process handlers volunteering for servicing the processing job based on its availability for servicing the processing job; and

wherein the communicating step comprises the request handler communicating data for the processing job to the process handler that volunteered for servicing the processing job.

72. A method for processing information, the method comprising:

receiving a plurality of service requests for a plurality of processing jobs, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow; and

executing the processing jobs in a distributed manner by a plurality of networked computers and in accordance with the received service requests, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, wherein the executing step comprises:

the request handler storing state information for the processing jobs;

the request handler communicating data relating to the processing jobs to a plurality of the process handlers;

the process handlers to which the data relating to the processing jobs were communicated (1) analyzing the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identifying a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determining that the processing job corresponding to the process flow with no remaining processing tasks has been completed;

the task handlers performing the identified processing tasks to generate a plurality of task results; and

updating the stored state information based on the task results.

73. The method of claim **72** wherein the executing step further comprises:

the process handlers volunteering for servicing the processing jobs based on their availabilities for servicing the processing jobs.

74. The method of claim **73** wherein the executing step further comprises:

the request handler (1) selecting the process handlers for servicing the processing jobs from among the process handlers that volunteered for same, and (2) communicating the data relating the processing jobs to the selected process handlers.

75. The method of claim **74** wherein the executing step further comprises:

the selected process handlers communicating a plurality of task requests for the identified processing tasks to a plurality of the task handlers;

the tasks handlers to which the task requests were communicated volunteering for performing the identified tasks corresponding to the task requests based on their being able to perform the identified tasks corresponding to the task requests; and

the selected process handlers selecting the task handlers for performing the identified tasks corresponding to the task requests from among the task handlers that volunteered for same.

76. The method of claim **74** wherein the executing step further comprises:

the task handlers communicating updated state information for the processing jobs following completion of their identified processing tasks to the selected process handlers.

77. The method of claim **76** wherein the executing step further comprises:

the selected process handlers communicating updated state information for the processing jobs to the request handler.

78. The method of claim **73** wherein the executing step further comprises:

the request handler communicating a plurality of processing requests for the processing jobs to a plurality of the process handlers;

the process handlers to which the processing requests were communicated volunteering for servicing the processing jobs corresponding to the processing requests based on their availabilities for servicing the processing jobs corresponding to the processing requests; and

the request handler (1) selecting the process handlers for servicing the processing jobs corresponding to the processing requests from among the process handlers that volunteered for same, and (2) communicating the data relating to the processing jobs to the selected process handlers.

79. The method of claim **72** wherein the executing step further comprises:

the task handlers volunteering for performing the identified tasks based on their being able to perform the identified tasks.

80. The method of claim **79** wherein a plurality of the networked computers on which a plurality of the task handlers reside have different resources; and wherein the executing step further comprises:

a plurality of the task handlers volunteering for performing the identified tasks as a function of the resources of the networked computers on which the plurality of task handers are resident.

81. The method of claim **80** wherein the different resources comprise different specialized hardware.

82. The method of claim **79** wherein a plurality of the networked computers on which a plurality of the task handlers are resident are grouped into a plurality of territories, and wherein the executing step further comprises:

assigning at least one of the identified tasks to a task handler based at least in part on territory considerations.

83. The method of claim **82** wherein a plurality of the networked computers have an associated physical location, and wherein the plurality of the networked computers having an associated physical location are grouped into the territories based on their associated physical locations such that each territory comprises a plurality of networked computers having an associated physical location within that territory.

84. The method of claim **72** wherein a plurality of the networked computers on which a plurality of the task handlers reside have different resources, and wherein the executing step further comprises:

assigning a plurality of the identified tasks to a plurality of the task handlers based at least in part on the resources of the networked computers on which the plurality of task handers are resident.

85. The method of claim **84** wherein the assigning step further comprises:

assigning at least one of the identified tasks to a task handler in order to limit a number of accesses to a database engine that will be needed to perform the at least one assigned task.

86. The method of claim **84** wherein the assigning step further comprises:

assigning at least one of the assigned tasks to a task handler in order to limit a rate of access to a database engine that will be needed to perform the at least one assigned task.

87. The method of claim **84** wherein the assigning step further comprises:

assigning at least one of the assigned tasks to a task handler in order to limit a number of accesses to a storage device that will be needed to perform the at least one assigned task.

88. The method of claim **84** wherein the assigning step further comprises:

assigning at least one of the assigned tasks to a task handler in order to limit a rate of access to a storage device that will be needed to perform the at least one assigned task.

89. The method of claim **84** wherein the different resources comprise different specialized hardware.

90. The method of claim **84** wherein the assigned tasks comprise limited tasks.

91. The method of claim **84** wherein the executing step further comprises:

selecting a task handler to which an identified task will be assigned from among a plurality of the task handlers based at least in part on the resources of the networked computers on which the a plurality of the task handers are resident.

92. The method of claim **91** wherein the task handler selecting step comprises a process handler to which data relating to a processing job was communicated selecting the task handler to which an identified task will be assigned from among a plurality of the task handlers based at least in part on the resources of the networked computers on which the a plurality of the task handers are resident.

93. The method of claim **92** wherein the at least one of the networked computers has a process handler and a task handler resident therein.

94. The method of claim **84** wherein the executing step further comprises:

selecting a task handler to which an identified task will be assigned from among a plurality of the task handlers based at least in part on load balancing considerations.

95. The method of claim **72** wherein a plurality of the networked computers on which a plurality of the task handlers are resident are grouped into a plurality of territories, and wherein the executing step further comprises:

assigning at least one of the identified tasks to a task handler based at least in part on territory considerations.

96. The method of claim **95** wherein a plurality of the networked computers have an associated physical location, and wherein the plurality of the networked computers having an associated physical location are grouped into the territories based on their associated physical locations such that each territory comprises a plurality of networked computers having an associated physical location within that territory.

97. The method of claim **72** wherein the executing step further comprises:

the request handler (1) receiving at least one of the service requests from a client computer, and (2) communicating a processing result for the processing job corresponding to the at least one service request to the client computer.

98. The method of claim **97** further comprising the client computer communicating via a network with the networked computer on which the request handler is resident.

99. The method of claim **72** wherein the request handler is resident on a first of the networked computers, wherein at least one of the process handlers is resident on a second of the networked computers, and wherein at least one of the task handlers is resident on a third of the networked computers.

100. The method of claim **72** wherein the request handler and at least one of the process handlers are resident on the same one of the networked computers.

101. The method of claim **72** wherein the request handler is resident on a different one of the networked computers than the networked computers on which the process handlers and task handlers are resident.

102. The method of claim **101** wherein the process handlers are resident on different ones of the networked computers than the networked computers on which the task handlers are resident.

103. The method of claim **72** wherein at least one of the process handlers and at least one of the task handlers are resident on the same one of the networked computers, the at least one task handler being on a different thread of the networked computer than the at least one process handler.

104. The method of claim **72** wherein the logic for a process flow comprises (1) a plurality of state variables before and after each processing task of that process flow, the state variables configured to store the state information, and (2) a plurality of transitions from state to state based on results for the processing tasks of that process flow.

105. The method of claim **104** wherein the executing step further comprises:

the networked computers processing the processing jobs according to a plurality of process definition files corresponding to the processing jobs, each process definition file defining the process flow for a processing job.

106. The method of claim **72** wherein at least one of the process flows further includes a recovery procedure for the processing tasks of the at least one process flow.

107. The method of claim **72** wherein the task results include updated state information for the processing jobs; and

wherein the updating step comprises the request handler storing the updated state information based on the updated state information included in the task results.

108. The method of claim **75** wherein the step of the selected process handlers communicating the task requests to the task handlers comprises the selected process handlers sending the task requests to the task handlers.

109. The method of claim **73** wherein the executing step further comprises:

the tasks handlers volunteering for performing the identified tasks based on their being able to perform the identified tasks.

110. The method of claim **109** wherein the executing step further comprises:

a plurality of the process handlers communicating a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

111. The method of claim **73** further comprising:

the request handler assigning global unique identifiers (GUIDs) to the processing jobs.

112. The method of claim **111** wherein the receiving step comprises the request handler receiving at least one of the processing job service requests from a client computer, the method further comprising:

the request handler providing the client computer with the GUID assigned to that processing job.

113. The method of claim **111** wherein the executing step further comprises:

the request handler determining whether a recovery procedure is to be initiated for a processing job; and

in response to a determination that a recovery procedure is to be initiated for a processing job, the request handler generating a recovery request message that includes the GUID for that processing job.

114. The method of claim **113** wherein the networked computers further comprise a recovery handler, and wherein the executing step further comprises:

the request handler communicating the generated recovery request message to the recovery handler.

115. The method of claim **72** wherein the processing jobs define a plurality of transactions.

116. The method of claim **115** wherein the transactions comprise a plurality of independent transactions.

117. The method of claim **115** wherein the transactions comprise a plurality of credit card processing transactions.

118. The method of claim **73** wherein each processing job is associated with a data structure that defines the process flow, the data structure including data that identifies a plurality of files for carrying out tasks, each file associated with a task.

119. The method of claim **118** wherein each of a plurality of the process flow data structures further comprises:

a plurality of state identifiers for different states of the process flow;

a plurality of task identifiers for different tasks of the process flow, each task identifier being associated with a state identifier; and

a plurality of associations between the state identifiers and a plurality of potential outcomes of the tasks corresponding to the task identifiers.

120. The method of claim **119** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of retry count limits, each retry count limit data being associated with a task identifier to define how many retries are to be performed for the associated task before an error condition is found.

121. The method of claim **119** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of timeout count limits, each timeout count limit data being associated with a task identifier to define how many timeouts are to be permitted for the associated task before an error condition is found.

122. The method of claim **118** wherein the executing step further comprises:

the tasks handlers volunteering for performing the identified tasks based on their being able to perform the identified tasks; and

a plurality of the process handlers communicating a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

123. A system for processing information, the system comprising:

a plurality of networked computers for processing a plurality of processing jobs in a distributed manner, the plurality of networked computers comprising a request handler, a plurality of process handlers, and a plurality of task handlers, the process handlers being resident on a plurality of different networked computers, the task handlers being resident on a plurality of different networked computers, the processing jobs having a plurality of associated process flows, the process flows including (1) a plurality of processing tasks and (2) logic configured to define a relationship between the processing tasks of the same process flow;

wherein the request handler is configured to (1) receive a plurality of service requests for the processing jobs, and (2) store state information for the processing jobs;

wherein the process handlers are configured to volunteer for servicing the processing jobs based on their availabilities;

wherein the request handler is further configured to communicate data relating to the processing jobs to a plurality of the process handlers that volunteered;

wherein the process handlers to which the data relating to the processing jobs were communicated are configured to (1) analyze the state information for the processing jobs to determine whether any processing tasks in the process flows remain to be performed based on the logic for the process flows, (2) in response to the state information analysis indicating that a processing task remains for the process flow of a processing job, identify a processing task to be performed for the process flow having the remaining processing task, and (3) in response to the state information analysis indicating that no processing tasks remain for the process flow of a processing job, determine that the processing job corresponding to the process flow with no remaining processing tasks has been completed;

wherein the task handlers are configured to volunteer for performing tasks based on their availabilities;

wherein a plurality of the task handlers that volunteered are configured to perform the identified processing tasks to generate a plurality of task results; and

wherein the request handler is further configured to store updated state information for the processing jobs, the updated stored state information being based on the task results.

124. The system of claim **123** wherein the request handler is further configured to communicate a plurality of processing requests for the processing jobs to a plurality of the process handlers;

wherein the process handlers to which the processing requests were communicated are configured to volunteer for servicing the processing jobs based on their availabilities in response to the processing requests;

wherein a plurality of the process handlers are further configured to communicate a plurality of task requests for the identified tasks to a plurality of the task handlers; and

wherein the task handlers to which the task requests were communicated are configured to volunteer for performing the identified tasks based on their availabilities.

125. The system of claim **124** wherein the request handler is further configured to select the process handlers for servicing the processing jobs from among the volunteering process handlers; and

wherein the process handlers are further configured to select the task handlers for performing the identified tasks from among the volunteering task handlers.

126. The system of claim **123** wherein the request handler is further configured to (1) initiate a state data structure associated with a processing job, the state data structure configured to store data indicative of a state for the associated processing job, (2) receive a plurality of updates regarding the state for the associated processing job, and (3) update the state data structure based on the received updates.

127. The system of claim **126** wherein the request handler is further configured to assign global unique identifiers (GUIDs) to the processing jobs.

128. The system of claim **127** wherein the request handler is further configured to (1) receive at least one of the processing job service requests from a client computer, and (2) provide the client computer with the GUID assigned to that processing job.

129. The system of claim **127** wherein the request handler is further configured to (1) determine whether a recovery procedure is to be initiated for a processing job, and (2) in response to a determination that a recovery procedure is to be initiated for a processing job, generate a recovery request message that includes the GUID for that processing job.

130. The system of claim **129** wherein the networked computers further comprise a recovery handler, and wherein the request handler is further configured to communicate the generated recovery request message to the recovery handler.

131. The system of claim **123** wherein the processing jobs define a plurality of transactions.

132. The system of claim **131** wherein the transactions comprise a plurality of independent transactions.

133. The system of claim **131** wherein the transactions comprise a plurality of credit card processing transactions.

134. The system of claim **123** wherein each processing job is associated with a data structure that defines the process flow, the data structure including data that identifies a plurality of files for carrying out tasks, each file associated with a task.

135. The system of claim **134** wherein each of a plurality of the process flow data structures further comprises:

a plurality of state identifiers for different states of the process flow;

a plurality of task identifiers for different tasks of the process flow, each task identifier being associated with a state identifier; and

a plurality of associations between the state identifiers and a plurality of potential outcomes of the tasks corresponding to the task identifiers.

136. The system of claim **135** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of retry count limits, each retry count limit data being associated with a task identifier to define how many retries are to be performed for the associated task before an error condition is found.

137. The system of claim **135** wherein each of the plurality of the process flow data structures further comprises:

data representative of a plurality of timeout count limits, each timeout count limit data being associated with a task identifier to define how many timeouts are to be permitted for the associated task before an error condition is found.

138. The system of claim **134** wherein a plurality of the process handlers are further configured to communicate a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

139. The system of claim **123** wherein a plurality of the process handlers are further configured to communicate a plurality of task requests for the identified tasks to a plurality of the task handlers that volunteered for performing those tasks, and wherein each task request is associated with a global unique identifier (GUID) therefor.

* * * * *

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a true and correct copy of the foregoing document was served on opposing counsel through their counsel of record via the Court's CM/ECF system on the 4th day of October, 2016.

R. William Sigler
Alan M. Fisch
Jennifer K. Robinson
Jeffrey M. Saltman
Fisch Sigler LLP
5301 Wisconsin Avenue NW, Fourth Floor
Washington, DC 20015
P. (202) 362-3500
F: (202) 362-3601
Bill.Sigler@Fischllp.com
Alan.Fisch@Fischllp.com
Jennifer.Robinson@Fischllp.com
Jeffrey.Saltman@Fischllp.com

David M. Saunders
Fisch Sigler LLP
96 North Third Street, Suite 260
San Jose, California 95112
David.Saunders@Fischllp.com

Attorneys for Amazon.com, Inc.
and Amazon Web Services, Inc.

/s/ Anthony G. Simon
Anthony G. Simon
Attorney for Plaintiff–Appellant

CERTIFICATE OF COMPLIANCE

1. The brief complies with the type-volume limitation of Federal Rule of Appellate Procedure 32(a)(7)(B). The brief contains 10,210 words, excluding the parts of the brief exempted by Federal Rule of Appellate Procedure 32(a)(7)(B)(iii) and Federal Circuit Rule 32(b).

2. The brief complies with the typeface requirements of Federal Rule of Appellate Procedure 32(a)(5) and the type style requirements of Federal Rule of Appellate Procedure 32(a)(6). The brief has been prepared in a proportionally-spaced typeface using Microsoft Word 2013 in Times New Roman 14-point font.

Dated: October 4, 2016

/s/ Anthony G. Simon
Anthony G. Simon
Attorney for Plaintiff-Appellant